ดิจิตอลลอจิกซิมมู เล เตอร์

Digital Logic Simulator

โดย

ชัยภัทร์ ปิยะดำรง

ดิษศรัย ปิณฑะดิษ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมอิเลคทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง

ปีการศึกษา 2535

ปีการศึกษา 2535

Digital Logic Simulator

โดย

1. นายชัยภัทร์ ปิยะดำรง 32.1072

2. นายดิษศรัย ปิณฑะดิษ 32.1096

อาจารย์ที่ปรึกษา

รศ.ดร.มนัส สังวรศิลป์

ปริญญานิพนธ์ปีการศึกษา 2535

ภาควิชาอิเลคทรอนิคส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง Digital Logic Simulator

ผู้จัดทำ

1. นายชัยภัทร์ ปิยะดำรง 32.1072

2. นายดิษศรัย ปิณทะดิษ 32.1096

_____อาจารย์ที่ปรึกษา

(รศ.ดร.มนัส สังวรศิลป์)

ดิจิตอลลอจิกซิมมู เล เตอร์

(Digital Logic Simulator)

นาย ชัยภัทร์ ปิยะดำรง 32.1072
นาย ดิษศรัย ปิณฑะดิษ 32.1096
อาจารย์ที่ปรึกษา:
รศ.ดร. มนัส สังวรศิลป์
เทอมที่ 2 ปีการศึกษา 2535

บทคัดย่อ

ในการสร้างวงจรทางดิจิตอลขึ้นมาใช้งานนั้น ขั้นแรกมีการออกแบบโดยทางทฤษฎี เมื่อได้วงจรแล้วก็จะนำเอาอุปกรณ์ต่างๆที่ใช้ในวงจร มาต่อเป็นวงจรเพื่อทำการทดสอบการทำงานว่าเป็นไปตามความต้องการหรือไม่ จากนั้นจึงนำวงจรที่ถูกต้องมาสร้างเป็นวงจรจริงในเชิงอุตสาหกรรม

ดิจิตอลลอจิกซิมมูเลเตอร์ (Digital Logic Simulator) เป็น ซอฟแวร์ที่เขียนขึ้นมา เพื่อใช้ในการทดสอบการทำงานของวงจร สามารถนำมาใช้แทนการทดลองต่อวงจรโดยทำงานในลักษณะที่มีการเลียนแบบ (Simulate) การทำงานของวงจรจริง สามารถป้อนอินพุทที่ใช้กับวงจร ผ่านขบวนการที่ใช้ในการ ซิมมูเลท ได้เอ้าท์พุทออกมา เป็นการสะดวกมากกว่า หากพบว่าวงจรที่ออกแบบมาแล้วนั้นไม่สามารถทำงานได้อย่างถูกต้อง โดยผู้ใช้สามารถแก้ไขดัดแปลงได้ง่ายกว่าการแก้ไขวงจรที่ต่อไปแล้ว

การทำงานของ ดิจิตอล ลอจิก ซิมมูเลเตอร์ นี้เป็นการทำงานในรูปแบบของซอฟแวร์ โดยผู้ใช้ต้องป้อนข้อมูลซึ่งในที่นี้หมายถึง รูปวงจร ให้กับ ซอฟแวร์และสร้างเงื่อนไขต่างๆที่มีการกำหนดรูปแบบโดยผู้เขียนไว้แล้ว โดยทั้งหมดนี้เป็นไปในรูปแบบของ เท็กไฟล์ ทำการ รัน (Run) โปรแกรมตามขั้นตอนต่างๆ โดยในท้ายที่สุดผลที่ได้จะอยู่ในรูปของไทม์มิงไดอะแกรม แสดงการทำงานของวงจร

ผลที่ได้จากการซิมมูเลทนั้นจะเหมือนกับผลที่ได้จากการต่อวงจรจริง ดังนั้น เมื่อ ไทม์มิงไดอะแกรม แสดงการทำงานได้อย่างถูกต้องตามความต้องการแล้ววงจรนั้นก็พร้อมที่จะนำไปสร้างวงจรจริงในทางปฏิบัติได้โดยไม่เกิดข้อผิดพลาด

Digital Logic Simulator

Mr.Chaiyapat Piyadamrong 32.1072
Mr.Dissai Pindatisha 32.1096
Advisor:
Dr.Manas Sangworasil
2nd Semester, 1992

Abstract

To make Digital Circuit for applying. The first step is to design theoretically and when the circuit is finished, putting device used in circuit together for testing whether it works as our desirability. The next step is that correct circuit passed testing can be transformed to real circuit for producing in industrial line.

Digital Logic Simulator is the software written to test functions of circuit. It can replace a making of real circuit by simulating from it. We can enter the input into simulating procedure for product, output which is more convenience to user adapting and adjusting designed circuit than performs on the already - created circuit.

The working of Digital Logic Simulator is software - performanced form which user have to enter data (it means circuit) to software and any condition definited already by programmer. The whole things are in TEXT FILE form. Running the program according to procedures which the final consequent timing diagram representing the circuit working.

The result from simulation resembles of testing real circuit therefore when timing diagram has shown properly working as our requirement, thai circuit is ready to be created in practice for real circuit without error.

สารบัญ

หน้า

สารบัญรูป

หน้า

บทนำ

ในช่วงระยะเวลาไม่นานมานี้ เทคโนโลยีทางด้านอิเลคทรอนิคส์เจริญขึ้นจนแทบจะตามไม่ทัน การกำเนิดของไมโครโปรเซสเซอร์ทำให้เกิดการเปลี่ยนแปลงครั้งยิ่งใหญ่ในวงการอิเลคทรอนิคส์จากวงจรอิเลคทรอนิกส์ซึ่งเป็น ฮาร์ดแวร์ มาเป็นวงจร ไมโคร-โปรเซสเซอร์ ซึ่งมีทั้ง ฮาร์ดแวร์ และ ซอฟต์แวร์ ซึ่งในปัจจุบันนี้เครื่องมืออิเลคทรอนิกส์เกือบทุกชนิดจะมี ไมโครโปรเซสเซอร์ อยู่ภายใน

การพัฒนาเทคโนโลยีในการผลิตวงจรรวม (Integrated Circuit-IC) ซึ่งประกอบด้วย ทรานซิสเตอร์ และองค์ประกอบวงจรต่างๆ ก็เปลี่ยนมาใช้วงจรรวมแทน และ ตัวอย่างวงจรที่เรียกซับซ้อนมากอย่างเช่น นาฬิกา, ดิจิตอลมิเตอร์ ก็เหลือเป็นเพียง LSI เพียงชิพเดียวกับอุปกรณ์ประกอบเล็กๆ อีกเพียงไม่กี่ชิ้นเท่านั้น

การศึกษาวงจรทางอิเลคทรอนิกส์กำลังเกิดการเปลี่ยนแปลงไปอย่างมาก จากการเรียนรู้เทคนิคในการต่อวงจรด้วยอุปกรณ์เล็กๆ มาเป็นการเรียนรู้วิธีการใช้วงจรรวมหรือ LSI ให้ถูกต้อง ซึ่งภายใน LSI ประกอบด้วยอุปกรณ์ต่างๆเป็นจำนวนมาก การที่จะสร้างหรือศึกษาการทำงานของวงจร จำเป็นต้องนำอุปกรณ์จำนวนมากมาทดลองต่อกันเอง ทำให้เสียเวลาและสิ้นเปลืองโดยใช่เหตุ

จากเหตุผลต่างๆดังกล่าวข้างต้นเป็นเหตุให้เกิดมีการพัฒนาการเขียนโปรแกรมจำลองการทำงานทั้งหมดของวงจรต่างๆโดยที่มุ่งเน้นไปที่วงจรทางลอจิก ซึ่งจะมีประโยชน์อย่างมากในการจำลองการทำงานของ CMOS และ TTL

บทที่ 1 ลักษณะทั่วไปของ Digital Logic Simulation

1.1 Digital Logic Simulation

Digital Logic Simulator หมายถึง อุปกรณ์ที่ใช้ในการ simulate การทำงานของ Digital circuit สามารถนำมาใช้แทนการต่อวงจรเมื่อทดลองจริงๆ

การออกแบบวงจร Digital เพื่อนำมาใช้งานนั้น ขั้นตอนต่างๆต้องมีทั้งการออกแบบทฤษฎี และนำผลที่ได้มาทดสอบทางปฏิบัติ ซึ่งในทางปฏิบัตินั้นถ้าหากว่าต้องทำการทดลองต่อวงจร และทำการป้อนข้อมูล เพื่อวิเคราะห์หา output ของมัน จะเห็นได้ว่าเป็นการไม่สะดวกหากวงจรที่ออกแบบมาแล้วนั้น ไม่สามารถทำงานได้ output ออกมาตรงกับความต้องการ

ดังนั้นจึงมีการคิดค้นอุปกรณ์บางอย่าง เพื่อนำมาใช้แทนการทดลองต่อวงจรจริง การนำขบวนการทาง software มาใช้จึงเป็นสิ่งที่เหมาะสม Digital Logic Simulator เป็น software ที่เขียนขึ้นมาเพื่อใช้งานแทนขั้นตอนในการทดลองต่อวงจรจริง โดยในทางปฏิบัติสิ่งที่จำเป็นคือ ผู้ใช้ต้องบอกตัว software ให้รู้ว่าวงจรเป็นอย่างไร ในปัจจุบันนี้วิธีการ Simulate โดยใช้ software นี้มีอยู่ 2 วิธีแตกต่างกันตรงวิธีการป้อนวงจรให้กับ software คือวิธีแรกที่จะเป็นการป้อนโดยใช้รูป โดย software จะมี program ย่อยทำหน้าที่ช่วยในการรับรูปวงจรจากผู้ใช้โดยการวาด program Simulate ที่ทำงานอย่างนี้ก็เช่น Orcad VSI อีกวิธีหนึ่งคือ ป้อนวงจรโดยให้ผู้ใช้เขียนเป็น Text File โดยเขียนตามรูปแบบที่ผู้เขียนกำหนดเท่านั้น ถ้าผิดจากรูปแบบไป เราเรียกว่าเกิด Syntax Error

1.2 ส่วนประกอบ

Digital Logic Simulator เป็น software ที่เขียนขึ้นมาใช้งาน เนื่องจากการ Simulate มีขั้นตอนมาก ดังนั้นในการเขียน program ผู้เขียนจึงแบ่งเป็นงานย่อยเพื่อสะดวกในการแก้ไขและพัฒนา เมื่อการทำงานแยกส่วนที่ต้องทำได้แล้วนำแต่ละงานย่อยมาเขียน program แต่ละตัวอีกที

โปรแกรมย่อยที่ถูกแบ่งออกมาแล้วมีจำนวน 3โปรแกรม สามารถเขียนและ Compile ออกมาเป็น execute file ได้ 3 file คือ LCT.EXE, LCS.EXE และ LCG.EXE ในการ Compile จาก source file จำพวก LCD File, SIM File และ DAT File ให้เสร็จขบวนการจนได้ output ออกมาในรูปของ Timing Diagram ยังจะต้องใช้file อื่นๆในการช่วยให้ขบวนการ Compile สำเร็จ

File ที่ต้องใช้ทั้งหมด สามารถแบ่งแยกตามหน้าที่การทำงานได้ดังนี้

-ExecuteFile ที่ใช้ในการแปลง source file เช่น LCT.EXE, LCS.EXE และ LCG.EXE

-Execute File ที่ใช้ในการ Compile เช่น TCC.EXE, TLINK.EXE, TLIB.EXE

-Library File ที่เป็น source file เช่น GATE.LCD, JKFF.LCD เป็นต้น

-Library File ที่ใช้ในการ Compile เช่น LCSS.LIB, GRAPHIC.LIB CS.LIB เป็นต้น

-Include File ที่ใช้ในการ Compile เช่น Stdic.h, Stdarg.h, Time.h, Error.h เป็นต้น

-Object File ที่ใช้ในการ Compile เช่น COS.OBJ, COM.OBJ เป็นต้น

ไฟล์เหล่านี้ถูกรวบรวมไว้ใน disk และภายใน disk ยังต้องมีที่ว่างพอให้สร้าง Temporary File เพื่อใช้ในการ Compile

1.3 ขอบเขตในการใช้งาน

การทำงานส่วนใหญ่ของ Digital Logic Simulator ส่วนใหญ่เป็นผลมาจากการป้อนคำสั่งโดยการสั่งพิมพ์เข้าไป ดังนั้นคำสั่งที่ป้อนเข้าไปจึงต้องถูกต้อง มิฉะนั้นแล้ว program จะไม่ทำงาน และแจ้ง Error ออกมาเพื่อบอกให้ผู้ใช้ทราบว่าเกิด error ขึ้น เพราะส่วนใดและอย่างไร

การใช้งานควรมี editor เพื่อเพิ่มความสะดวกในการใช้งาน เพราะเมื่อเกิด Error ที่สาเหตุมาจากโปรแกรมพวก Source File เช่น LCT File, SIM File และ DAT File มีการเขียนผิดผู้ใช้จำเป็นต้องแก้ไข source file เหล่านั้นโดย load ไฟล์ที่ผิดเข้ามาแก้ใน editor

บทที่ 2 โครงสร้างของระบบ

การทำงานของ Digital Logic Simulator สามารถอธิบายได้ด้วยบล็อกไดอะแกรมดังแสดงในรูปที่ 2.1 จากรูปจะเห็นได้ว่าในการใช้งานต้องเขียนไฟล์ขึ้นมา 3 ไฟล์ด้วยกัน คือ LCD ไฟล์, SIM ไฟล์, DAT ไฟล์ เมื่อเริ่มใช้งานต้องใช้โปรแกรม LCT (LCD to C Translator) ในการแปลง LCD ไฟล์ ให้เป็น source code ในรูปของภาษาซี เช่น *.C, *.H เป็นต้น SIM ไฟล์ ให้แปลงโดยใช้ LCS (LCD Simulation Code Translator) เอ้าท์พุทที่ได้เป็น source code ในภาษาซี เช่น EXE.C ขั้นต่อไปใช้ Compilerของภาษาแปลงทั้ง 2 ไฟล์ ให้เป็น object ไฟล์ (*.obj) แล้วทำการลิงค์ (link) ออบเจ็คไฟล์ทั้ง 2 ไฟล์ เข้าด้วยกันจนได้เป็น เอ็กซิคิวท์ไฟล์ (*.exe) จากนั้นจะนำสัญญาณอินพุทที่ต้องการจะทดสอบ คือ DAT ไฟล์ เข้ามาร่วมในการประมวลผลจนได้เอ้าท์พุทออกมา ซึ่งยังไม่สะดวกในการวิเคราะห์ เนื่องจากอยู่ในรูปของไบนารี่โค้ด (binary code) ในขั้นตอนสุดท้ายส่วนที่เรียกว่า LCG (LCD Graphichs Utility) ซึ่งมีหน้าที่ในการแปลงจากรูปแบบของไบนารี่โค้ดให้เป็นแผนภูมิเวลา (Timing Diagram) โดยที่แต่ละไฟล์ที่ต้องเขียนขึ้นมีโครงสร้างและรูปแบบที่แน่นอนดังต่อไปนี้

2.1 LCD ไฟล์

LCD ไฟล์ (Logic Circuit Description File) เป็นไฟล์ที่เขียนบรรยายส่วนประกอบต่างๆของวงจรซึ่งจะกล่าวถึงชื่อวงจร, จุดต่อสัญญาณเข้า-ออก และฟังก์ชั่นการทำงานภายในวงจร โดยมีรูปแบบดังนี้

```
NAME      xxxx;        ชื่อวง
INPUT     xxxx;        จุดต่อสัญญาณทางเข้า
OUTPUT    xxxx;        จุดต่อสัญญาณทางออก
DEFINE                 กำหนดฟังก์ชั่นที่ใช้ในโปรแกรม
{
    EXT   xx;          ชื่อโปรแกรมภายนอกที่เรียกใช้
    LOC   xx;          ชื่ออุปกรณ์ภายในที่เรียกใช้โปรแกรมย่อยภายนอก
```

```
    U1 = FUNC x,x);   ฟังก์ชั่นของอุปกรณ์ภายใน
    G1 = A & B;
    :
    :
    :
}
END;                  สิ้นสุดไฟล์
```

2.2 SIM ไฟล์

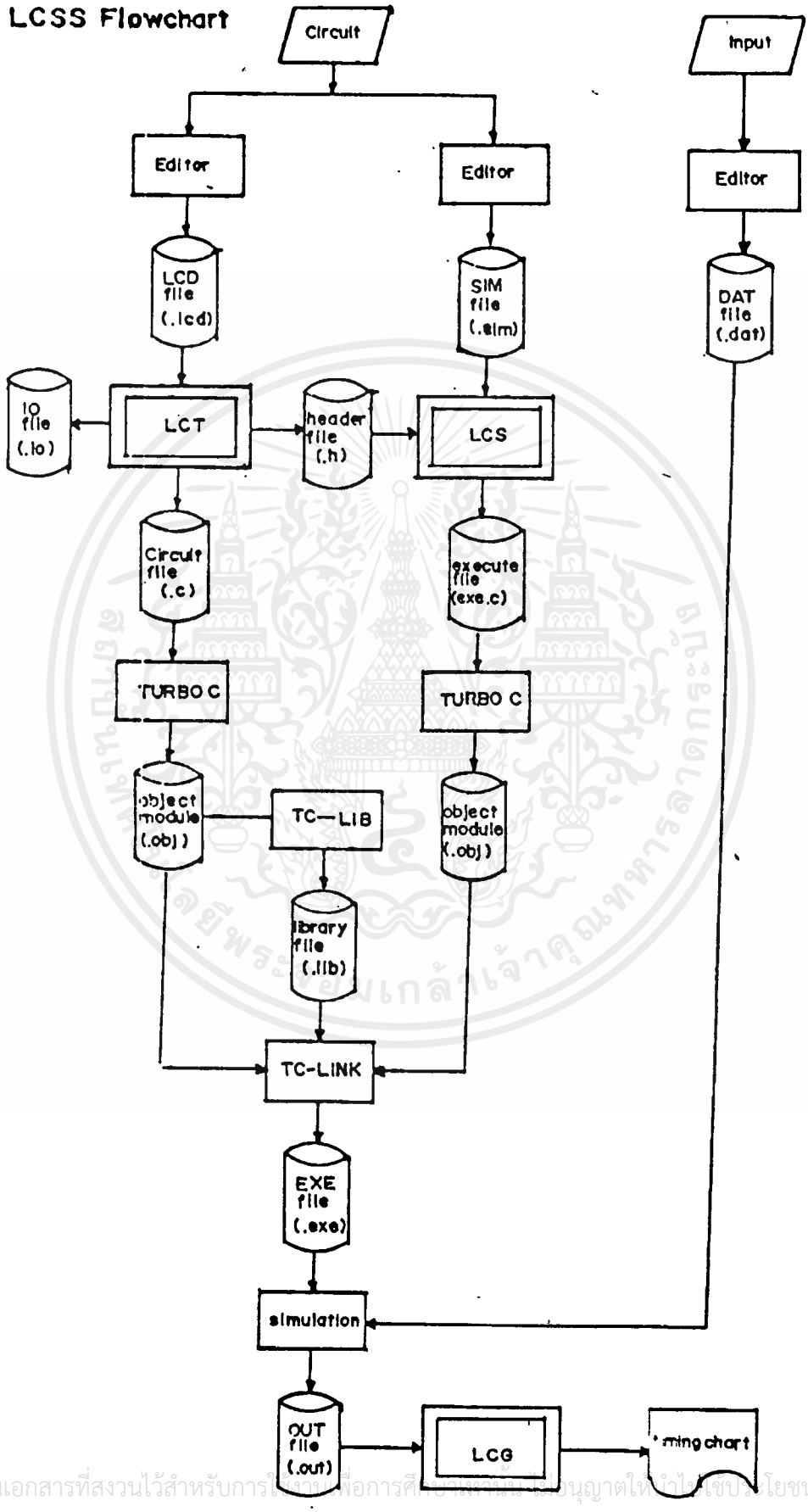
SIM ไฟล์ (Simulation File) เป็นไฟล์ที่ขึ้นเพื่อกำหนดจุดทางเข้า-ออกของสัญญาณที่ใช้ทดสอบ โดยมีรูปแบบดังนี้

```
TEST      xxxx;     ชื่อวงจร
TESTIN    xxxx;     จุดทางเข้าของสัญญาณที่ใช้ทดสอบ
TESTOUT   xxxx;     จุดทางออกของสัญญาณที่จะแสดงผล
END;                สิ้นสุดไฟล์
```

2.3 DAT ไฟล์

DAT ไฟล์ (Data File) เป็นไฟล์ที่เขียนขึ้นเพื่อกำหนดรูปแบบของสัญญาณที่ใช้ทดสอบ โดยมีรูปแบบดังนี้

```
PATTERN             รูปแบบของข้อมูลที่ใช้ทดสอบ
{
    0000 : xx       ค่าของข้อมูลชุดที่หนึ่ง
    0010 : xx       ค่าของข้อมูลชุดที่สอง
    :
}
END;                จบชุดข้อมูล
```



รูปที่ 2.1 บล็อกไดอะแกรมของระบบจำลองการทำงานของวงจรลอจิก

บทที่ 3 หลักการทำงาน

เราสามารถแบ่งการทำงานของ Digital Logic Simulator เป็นโปรแกรมย่อยได้ 3 โปรแกรม ซึ่งแต่ละโปรแกรมมีหลักการทำงานที่แตกต่างกันดังนี้

3.1 LCT (LCD To C Translator)

ส่วนนี้มีหน้าที่หลักในการแปลง LCD ไฟล์ ที่เขียนขึ้นเพื่อบรรยายส่วนประกอบของวงจร โดยใช้เอดิเตอร์ต่างๆให้เป็นภาษาซี ขั้นตอนการทำงานของ LCT ดังแสดงในรูปที่ 3.1 สามารถอธิบายได้ดังต่อไปนี้

จาก LCDไฟล์ ที่เขียนไว้ สแกนโปรเซส (scan process) จะทำการอ่านข้อมูล แล้วส่งต่อไปให้ส่วน พาร์ทโปรเซส (part process) และเทเบิลโปรเซส (table process) ทำการตีความหมาย ซึ่งจะมีการตรวจจับความผิดพลาดด้วยส่วนที่เรียกว่าเออร์เรอร์โปรเซส (error process) จากนั้นจะถูกส่งไปยังส่วน โคดโปรเซส (code process) เพื่อสร้างเป็นเอ้าท์พุท-อินพุทไฟล์ (I/O file), เฮดเดอร์ ไฟล์ (header file) และซอร์สไฟล์ (source file) ในรูปแบบของภาษาซี (*.C)

3.2 LCS (LCD Simulation Code Generator)

ส่วนนี้มีหน้าที่หลักในการแปลง SIM ไฟล์ ที่เขียนขึ้นเพื่อบรรยายส่วนประกอบของวงจร ซึ่งจะกล่าวถึง ชื่อวงจร, จุดต่อสัญญาณทางเข้า-ทางออก และฟังก์ชั่นการทำงานภายในวงจรโดยใช้เอดิเตอร์ต่างๆให้เป็นภาษาซี ขั้นตอนการทำงานของ LCS แสดงไว้ในรูปที่ 3.2 สามารถอธิบายได้ดังต่อไปนี้

จาก SIMไฟล์ ที่เขียนไว้ สแกนโปรเซส (scan process) จะทำการอ่านข้อมูล แล้วส่งต่อไปให้ส่วน พาร์ทโปรเซส (part process) และเทเบิลโปรเซส (table process) ทำการตีความหมาย ซึ่งจะมีการตรวจจับความผิดพลาดด้วยส่วนที่เรียกว่า เออร์เรอร์โปรเซส (error process) จากนั้นจะถูกส่งไปยังส่วน โคดโปรเซส (code process) เพื่อสร้างเป็นซอร์สไฟล์ (source file) อีกอันหนึ่ง (EXE.C)

3.3 LCG (LCD Graphic Utility)

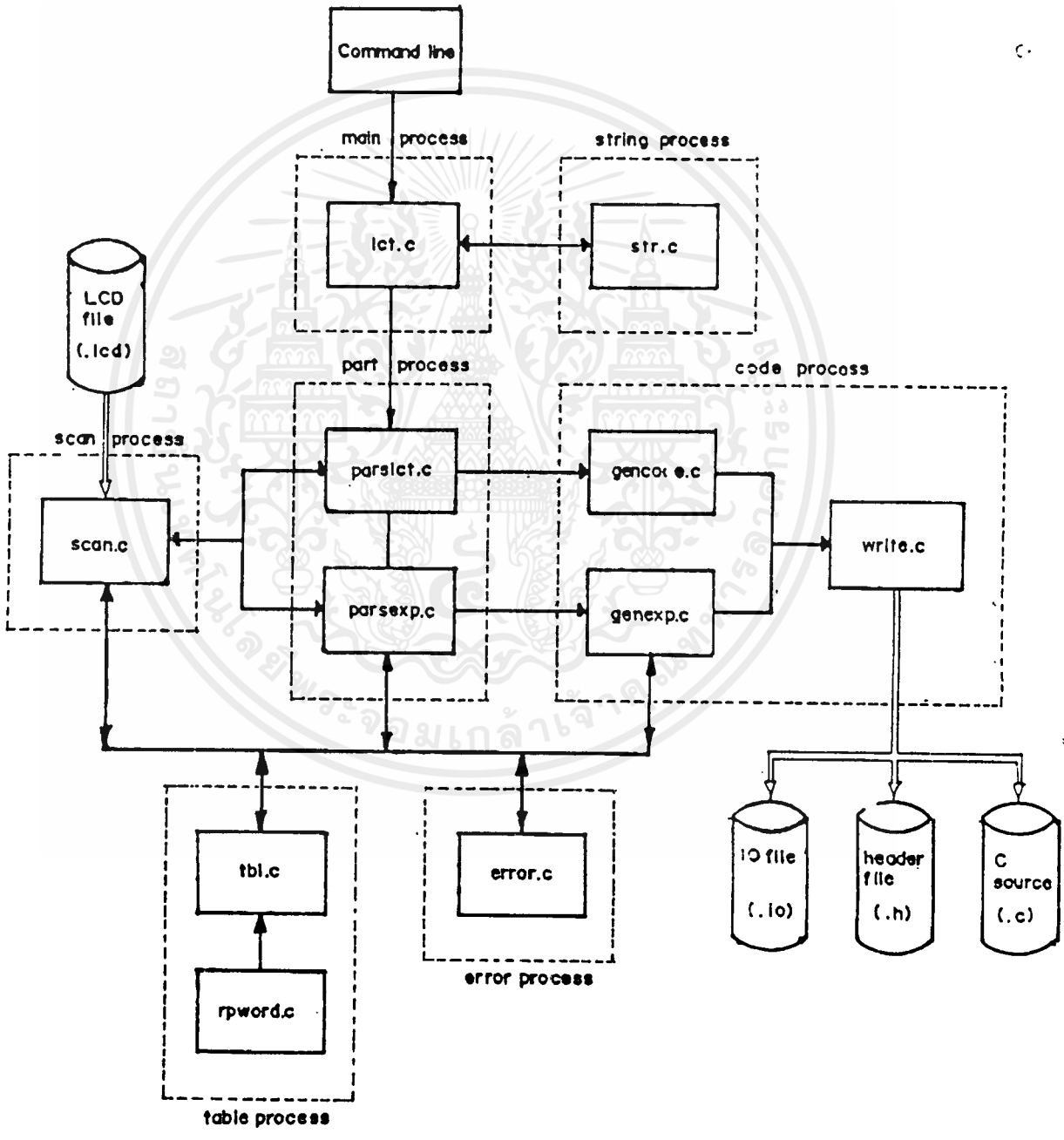
ส่วนนี้มีหน้าที่หลักในการแปลง OUT ไฟล์ ที่ได้จากการนำสัญญาณอินพุทเข้ามาร่วมในการประมวลผลของวงจร จนได้เอ้าท์พุทซึ่งจะนำไปแสดงให้เป็นแผนภูมิเวลา (timing diagram) เพื่อความสะดวกในการวิเคราะห์การทำงานของวงจร ขั้นตอนการทำงานของ LCG แสดงไว้ในรูปที่ 3.3 สามารถอธิบายได้ดังต่อไปนี้

จาก OUT ไฟล์ ที่เขียนไว้ สแกนโปรเซส (scan process) จะทำการอ่านข้อมูล แล้วส่งต่อไปให้ส่วน พาร์ทโปรเซส (part process) และเทเบิลโปรเซส (table process) ทำการตีความหมาย ซึ่งจะมีการตรวจจับความผิดพลาดด้วยส่วนที่เรียกว่า เออร์เรอร์โปรเซส (error process) จากนั้นจะถูกส่งไปยังส่วนกราฟโปรเซส (graph process) เพื่อสร้างเป็นแผนภูมิเวลา (Timing Diagram)

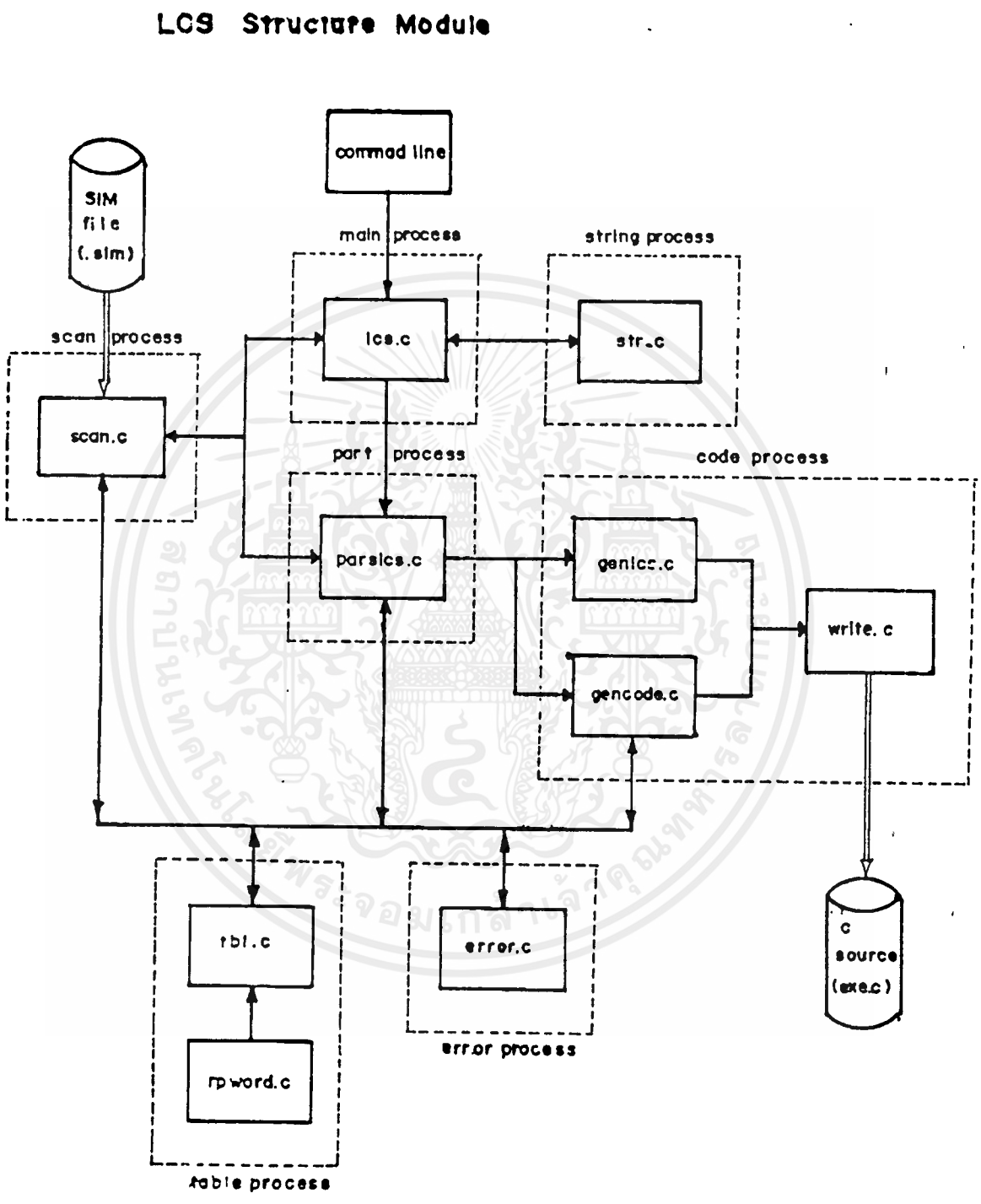
3.4 การทำงานของวงจร Compile

ยังมีอีกส่วนหนึ่งที่ยังไม่ได้กล่าวไว้ในส่วนหลัก3ส่วน แต่มีความสำคัญมากคือ การคอมไพล์ซอร์สไฟล์ทั้งสอง (*.C และ EXE.C) และทำการลิงค์ออบเจ็คไฟล์ทั้งสองเข้าด้วยกันเป็นเอ็กซิคิวท์ไฟล์ ซึ่งสามารถนำไปประมวลผลการทำงานโดยป้อนอินพุทแบบต่างๆ เข้าไปจะขอเรียกส่วนนี้ว่า EXE โดยมีการทำงานของ EXE ซึ่งแสดงไว้ในรูปที่ 3.4 สามารถอธิบายได้ดังต่อไปนี้

จาก DAT ไฟล์ ที่เขียนไว้ สแกนโปรเซส (scan process) จะทำการอ่านข้อมูล แล้วส่งต่อไปให้ส่วน พาร์ทโปรเซส (part process) และเทเบิลโปรเซส (table process)ทำการตีความหมาย ซึ่งจะมีการตรวจจับความผิดพลาดด้วยส่วนที่เรียกว่า เออร์เรอร์โปรเซส (errorprocess) เช่นเดียวกับ LCT และ LCS แต่จะมีส่วนไทม์เมอร์คอนโทรล เพิ่มขึ้นมาเนื่องจากในการประมวลผล จะมีสัญญาณนาฬิกาเข้ามาเกี่ยวข้องด้วย ขณะเดียวกัน สแกนโปรเซล (scan process) จะส่งข้อมูลให้กับ คอนโทรล-โปรเซส (control process) ซึ่งทำงานร่วมกับ ไฟล์โปรเซส (file process) โดยถูก เมนโปรเซส (main process) ควบคุมอยู่จากนั้นจะได้ผลการประมวลผลสัญญาณออกมาเป็น เอ้าท์พุทไฟล์ (output file)

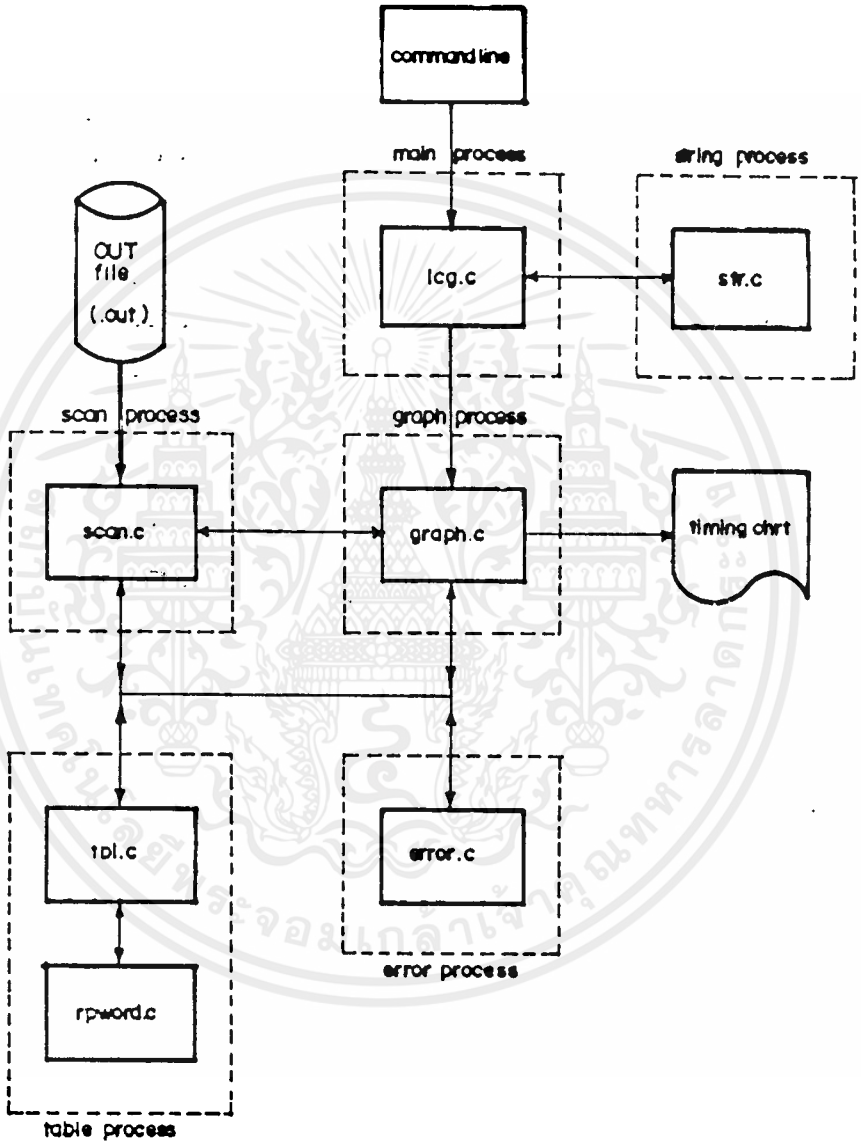


รูปที่ 3.1 บล็อกไดอะแกรมการทำงานของ LCT

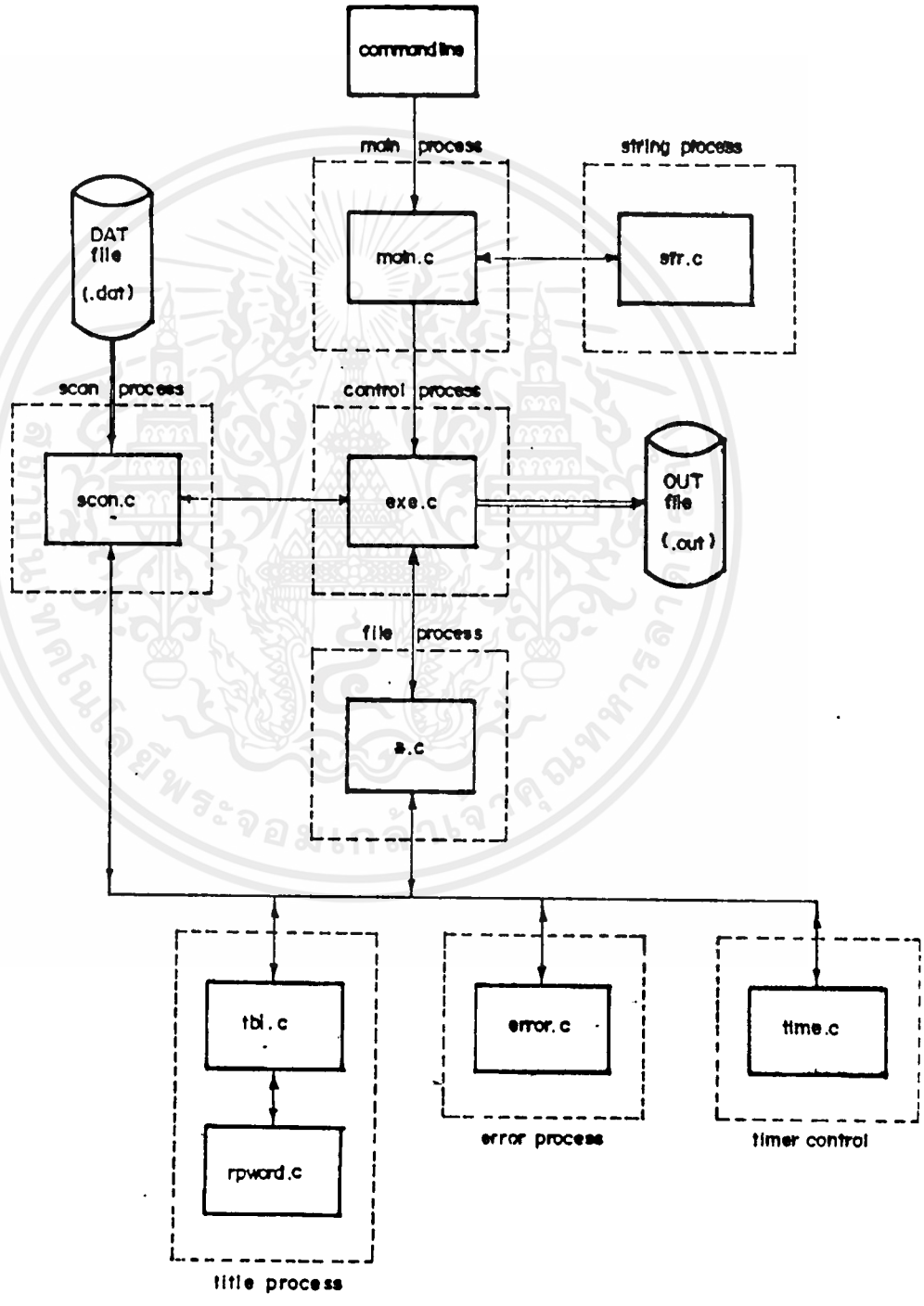


รูปที่ 3.2 บล็อกไดอะแกรมการทำงานของ LCS

LCG Structure Module



รูปที่ 3.3 บล็อกไดอะแกรมการทำงานของ EXE



รูปที่ 3.4 บล็อกไดอะแกรมการทำงานของ EXE

บทที่ 4 การใช้งาน

การใช้งาน Digital Logic Simulator สามารถแบ่งเป็นขั้นตอนสำคัญได้ดังนี้

4.1 การสร้าง Library

เริ่มจากการเขียนโปรมแกรมย่อยขึ้นมา 3 ตัว คือ LCT, LCS, LCG เพื่อทำหน้าที่ต่างกันดังที่กล่าวไว้แล้วในตอนต้น ในการ Simulate วงจรนั้น เนื่องจากวงจรที่นำมา Simulate นี้จะประกอบไปด้วยอุปกรณ์พื้นฐาน เช่น อุปกรณ์ Gate ต่างๆ เช่น AND GATE, OR GATE, NAND GATE เป็นต้น ซึ่งเป็นอุปกรณ์ที่ไม่มีความจำ หรือเป็นพวกอุปกรณ์ที่มีความจำ เช่น Register Flip-Flop ชนิดต่างๆ เช่น JK Flip-Flop, RS Flip-Flop

ในขั้นแรกได้เริ่มเขียน LCD ไฟล์ ของอุปกรณ์พื้นฐานก่อน โดยใช้พีชคณิตพื้นฐานเช่น AND, OR, NOT ซึ่งเป็น logic ที่มีอยู่แล้วในภาษาซี นำมาจัดเป็นสมการเลียนแบบการทำงานของอุปกรณ์แต่ละตัว เช่น AND GATE เนื่องจากภาษาซีการ and สำหรับ 2 expressions เข้าด้วยกัน เขียนด้วยสัญญาน ¦ ดังนั้นเมื่อแทนแต่ละ expression ด้วย A และ B โดยใช้ Y แทนผลลัพธ์ที่ได้จากการ operate 2 expressions จะได้

$$Y = A \mid B$$

จากสมการนี้ เมื่อเขียน (define) เป็นlibraryของ AND GATE 2 inputดังนั้นเมื่อมีการใช้library ตัวนี้ (มี Gate ตัวนี้ต่ออยู่ในวงจร) สมการจะสามารถ simulate ได้ output ที่สามารถแปรได้ตาม input ที่ขาของ Gate 2 ขานั้น

ต่อไปทำการเขียน library ของอุปกรณ์พื้นฐานตัวอื่นอีก เช่นOR GATE, NAND GATE, NOR GATE แล้วแต่ชนิดของ Gate เราสามารถแบ่งลงไปได้อีกตามจำนวน input ของมัน และตั้งชื่อเพื่อสร้างเป็น library ที่สามารถเรียกใช้ได้ต่อไป

เพื่อไม่ให้เป็นการเสียเวลาในการที่จะเรียก Library ของ Gate พื้นฐานแต่ละตัว ผู้เขียนได้สร้างLibrary ย่อยของ Gate เหล่านั้นรวมไว้ใน LCD File ไฟล์เดียวโดยให้ชื่อว่า GATE.LCD ซึ่งในอนาคตสามารถเขียนเพิ่มเติมเข้าไปได้เมื่อ Library ที่ต้องการเรียกใช้ยังไม่มีใน GATE.LCD รูปแบบของ Gate แต่ละตัวได้เป็น Text File ดังนี้

```
>TYPE GATE.LCD          ( ENTER )

NAME      AND2;
INPUT     A, B;
OUTPUT    Y;
DEFINE
{
     Y = A & B;
}
END;

NAME      AND3;
INPUT     A, B, C;
OUTPUT    Y;
DEFINE
{
     Y = A & B & C;
}
END;

NAME      AND4;
INPUT     A, B, C, D;
```

```
OUTPUT    Y;
DEFINE
{
     Y = A & B & C & D;
}
END;

NAME      AND5;
INPUT     A, B, C , D, E;
OUTPUT    Y;
DEFINE
{
     Y = A & B & C & D & E;
}
END;

NAME      AND6;
INPUT     A, B, C , D, E, F;
OUTPUT    Y;
DEFINE
{
     Y = A & B & C & D & E & F;
}
END;

NAME      ND2;
INPUT     A, B;
OUTPUT    Y;
```

```
DEFINE
{
     Y = ~(A & B);
}
END;

NAME      ND5;
INPUT      A,B,C,D,E;
OUTPUT    Y;
DEFINE
{
     Y  = ~(A & B & C & D & E);
}
END;

NAME      OR2;
INPUT     A, B;
OUTPUT    Y;
DEFINE
{
     Y = A ¦ B;
}
END;

NAME      OR3;
INPUT     A, B, C;
OUTPUT    Y;
DEFINE
```

```
{
     Y = A ¦ B ¦ C;
}
END;

NAME      OR4;
INPUT     A, B, C, D;
OUTPUT    Y;
DEFINE
{
     Y = A ¦ B ¦ C ¦ D;
}
END;

NAME      NOR2;
INPUT     A, B;
OUTPUT    Y;
DEFINE
{
     Y = ~(A ¦ B);
}
END;

NAME      EOR2;
INPUT     A, B;
OUTPUT    Y;
DEFINE
{
```

```
        Y = A ^ B;
    }
    END;

    NAME    NOT;
    INPUT   A;
    OUTPUT  Y;
    DEFINE
    {
        Y = ~A;
    }
    END;

    NAME    ND3;
    INPUT   A, B, C;
    OUTPUT  Y;
    DEFINE
    {
        Y = ~(A & B & C);
    }
    END;

    NAME    NOR3;
    INPUT   A, B, C;
    OUTPUT  Y;
    DEFINE
    {
        Y = ~(A ¦ B ¦ C);
```

```
}
END;
```

เมื่อมีการสร้าง Library ที่เป็นพื้นฐานเรียบร้อยแล้วเราจะมาพิจารณาถึงการเขียน Library ที่มีการเรียกใช้ Library พื้นฐานอีกที ซึ่ง Library พวกนี้เป็น Library ที่สามารถสร้างให้เป็นอุปกรณ์พวก Register หรือ Flip-Flop ได้ โดยหลักการแล้วคือการนำเอา Library ย่อยมาประกอบกันเป็นวงจรนั่นเอง หรือกล่าวอีกนัยหนึ่งคือวงจรประเภท Register Flip-Flop สร้างขึ้นโดยนำ Gate พื้นฐานเหล่านั้น มาต่อเป็นวงจรที่แตกต่างกัน ตามลักษณะการทำงานของ Flip-Flop ตัวนั้น

การทำงานของอุปกรณ์จำพวกนี้ มักมีสัญญาณนาฬิกาหรือ Clock มาควบคุมการทำงานในปริญญานิพนธ์นี้ผู้เขียนได้ทดลองเขียน JKFF, RSFF มาใช้งานโดยเป็นแบบที่มี Clock ด้วย ดังนั้นเมื่อมีการเรียกใช้ Flip-Flop เหล่านี้ จำเป็นที่ผู้ใช้ต้องเรียก Gate. LCD เนื่องจากส่วนประกอบ Flip-Flop มีการเรียกใช้ Gate พื้นฐานต่างๆ ในที่นี้มีการแสดงลักษณะการเขียน Library ของ JKFF ได้ดังนี้

```
>TYPE JKFF.LCD          ( ENTER )

NAME     JKFF;
INPUT    J, K, CK;
OUTPUT   Q, Q_;
DEFINE
{
     EXT   ND2,NOT;
     LOC   G1,G2,G3,G4,G5,G6,G7,G8,G9;

     G1 = ND3 (J,Q_,CK);
     G2 = NOT (CK);
```

```
    G3 = ND3 (K,Q,CK);
    G4 = ND2 (G1,G5);
    G5 = ND2 (G3,G4);
    G6 = ND2 (G4,G2);
    G7 = ND2 (G5,G2);
    G8 = ND2 (G6,G9);
    G9 = ND2 (G7,G8);
    Q  = G8;
    Q_ = G9;
}
END;
```

4.2 การสร้าง TEXT File ที่ใช้ในการชิมมูเลท

หลังจากที่ได้ Library ที่แทนการทำงานของ Gate พื้นฐานต่างๆและอุปกรณ์ Registor ที่สำคัญบางตัวแล้ว ผู้ใช้ต้องทำการสร้วง TEXT File 3 ไฟล์ ที่ใช้ในการชิมมูเลท โดยมีขั้นตอนดังนี้

4.2.1 การสร้าง LCD File จากในบทที่ 2.1 โครงสร้างของ LCD File เราจะทำการเขียน Text File เลียนแบบตามรูปแบบที่กำหนดไว้แล้ว เริ่มจาก

- NAME : มีการตั้งชื่อตามชื่อของ Digital Circuit ที่จะนำ Simulate โดยชื่อนี้จะเป็นชื่อเดียวกับชื่อของ Text File

- INPUT : ผู้ใช้จะต้องระบุขาของวงจรที่มีการ input เช่น FULL ADDER จะมี INPUT เป็น A,B,Carry In เป็นคนละตัวกัน

-OUTPUT : เป็นจุดที่ต้องการกำหนดให้เป็นทางสัญญานออกของ output terminal โดยแทนได้เช่นเดียวกับ Input

-DEFINE : เป็นการเริ่มต้นบอกให้ compiler รู้ว่าต่อไปนี้จะเป็นรูปแบบในการต่อวงจรโดยใน 2 บรรทัดถัดจาก define มาจะเป็น EXT และ LOC

-EXT : เป็นการบอกว่า Digital Circuit นี้ประกอบด้วย Gate พื้น

ฐานอะไรบ้างและ Library ที่เกี่ยวกับ Register ใดบ้าง

- LOC : เป็นตัวแปรที่จะใช้แทนอุปกรณ์ต่างๆในวงจร ส่วนใหญ่กำหนดให้ G1, G2,G3,... แทน Gate แต่ละตัวและ U1,U2,U3,... แทนอุปกรณ์ที่มี Library ของตัวมันเอง หรือเราจะกำหนดให้ ตัวแปรที่แทนอุปกรณ์นั้นเป็นชื่อเดียวกับ Library ของอุปกรณ์นั้นๆ เช่น ในวงจรมี JKFF อยู่ 2 ตัว อาจกำหนดได้เป็น JKFF1,JKFF2

หลังจากกำหนด EXT.LOC เรียบร้อยแล้ว ส่วนต่อไปจะเป็นการแปลงการต่อวงจรให้เป็นในรูปของตัวอักษร มีหลักดังนี้คือ

U = FUNC (x,x) ;

U : ตัวแปรที่ใช้แทนอุปกรณ์แต่ละตัวที่ถูกกำหนดไว้ใน LOC

FUNC : ชื่อของอุปกรณ์ตัวนั้นที่ถูกแทนด้วยตัวแปรที่ถูกกำหนดไว้ใน LOC

:ภายในวงเล็บบอกถึง "ขา" Input ของอุปกรณ์ว่าต่อมาจากขาใดของอุปกรณ์ตัวใด ในที่นี้มี 2 ชนิดคือ

1. มาจากอุปกรณ์ที่มี 1 output เท่านั้น สามารถแทน U ได้เลย

2.มาจากอุปกรณ์ที่มีมากกว่า 1 output ใช้ "." ในการแสดง

เช่น ขา Q_ ของ JK-FF แสดงได้ด้วย "JKFF.Q_"

ทำเช่นนี้ไปเรื่อยจนครบอุปกรณ์ทุกตัว โอกาสในการเกิด Error เมื่อทำการ Compile วงจรนี้คือ ในส่วนของ EXT ถ้าเรียกอุปกรณ์ที่วงจรใช้มาไม่ครบ หรือใน Loc มีการกำหนดประเภทตัวแปรที่ใช้ในวงจรมาไม่ครบหรือจะเป็นในส่วนของ FUNC เมื่อกำหนด Input ของ FUNC ตัวนั้นไม่ถูกต้อง เช่น Input เกิน หรือ ขาด ไม่ตรงกับจำนวนที่ต้องมี

แสดงการเขียน LCD File ของ FULL ADDER โดยมีการเรียกใช้ HALF ADDER, GATE Library ได้ดังนี้

```
>TYPE    FA.LCD          ( ENTER )

NAME     FA;
INPUT    Cl, A, B;
OUTPUT   C, S;
DEFINE
{
     EXT EOR2, HA;
     LOC G1, U1, U2;

     G1   = OR2 (U1.C, U2.C);
     U1   = HA (A, B);
     U2   = HA (Cl, U1.S);
     S    = U2.S;
     C    = G1;
}
END;
```

4.2.2 การสร้าง SIM File จากบทที่ 2.2 โครงสร้างของ SIM File เราจะทำการเขียน Text file ขึ้นมาอีก File หนึ่ง เพื่อใช้กำหนด Input และ Output ให้กับวงจร โดยโครงสร้างของ SIM File นี้มี 3 บรรทัด ดังนี้

- TEST : กำหนดชื่อของวงจรโดยพยายามใช้ชื่อเดียวกันทั้งหมดทุก File

-TESIN : กำหนด "ขา" ของวงจรว่า มีการป้อน Input เข้าที่ไหน อันดับการเรียงนี้จะมีผลกับ pattern ใน DAT File

- TESTOUT: กำหนด "ขา" ที่ต้องการให้ส่วนของ LCG Program แสดงสัญญาณ output ออกมาในรูปของ Timming Diagram

โอกาสเกิด Error ในการสร้างไฟล์นี้อาจมาจากสาเหตุดังนี้

- TESTIN มีตัวแปรตัวที่ไม่ได้กำหนดไว้ในส่วน Input ของ LCD File
- TESTOUT มีตัวแปรตัวที่ไม่ได้กำหนดไว้ในส่วน output ของ LCD File

แสดงการเขียน FA.SIM (SIM File Full Adder) บอกการป้อนสัญญาณ Input และต้องการดูสัญญาณOutput ที่ "ขา" ใดบ้าง ได้ดังนี้

4.2.3 การสร้าง DAT File จากบทที่ 2.4 โครงสร้างของ DAT File โดยในไฟล์นี้ ผู้ใช้สามารถกำหนด Input ที่ต้องการป้อนเพื่อใช้ในการ Simulate การเขียนไฟล์นี้ต้องเป็นไปตามรูปของ DAT File โดยกำหนดเป็นpattern มีค่าทั้งในแนวนอนและแนวตั้ง โดยแนวนอนจะเป็นข้อมูลที่ป้อนให้ ขา Input แต่ละขาโดยมีอันดับการเรียงตามอันดับในการเรียงของ TESTIN ใน SIM File จะเกิด error เมื่อจำนวน Input ที่ป้อนให้ไม่ตรงกับจำนวน Input ที่แสดงใน TESTIN ของ SIM File

ในแนวตั้ง คือ บรรทัดหนึ่งจะแทนเวลาที่เกิดขึ้น 1 ช่วงเวลาการทำงาน (1 cycle) เพราะฉะนั้นจำนวนบรรทัดจะไม่เป็นผลทำให้เกิดการ error ได้ แต่ในการใช้งานควรมีขนาดไม่เกิน 60 บรรทัด เพราะในส่วนของการแสดงผล Timing diagram มีความละเอียดในการแสดง Time ได้ประมาณ 60 cycles

แสดง pattern การป้อน input ของ Full Adder ใน FA.DAT ได้ดังนี้

```
>TYPE   FA.DAT          ( ENTER )

PATTERN
{
     : 0 0 0
     : 0 0 1
```

```
        : 0 1 0
        : 0 1 1
        : 1 0 0
        : 1 0 1
        : 1 1 0
        : 1 1 1
    }
END;
```

4.3 ขั้นตอนการใช้งาน

เมื่อได้ LCD ไฟล์, SIM ไฟล์, DATไฟล์ ที่มีรูปแบบตรงตามผังกำหนดแล้ว นำมาผ่านขบวนการมนการ Simulate ซึ่งแบ่งเป็นขั้นตอนได้ดังนี้

4.3.1 เริ่มจากทำการแปลง LCD ไฟล์ให้เป็นภาษาซี โดยใช้ LCT.EXE มีรูปแบบคำสั่งดังนี้

LCT (*.LCD) (LIB.LCD)

ในที่นี้ *.LCD หมายถึง LCD File ที่เขียนไว้แล้วของ Digital Circuit ที่จะทำการ Simulate หากว่า Digital Circuit นี้มีการเรียกอุปกรณ์อื่น จะต้องมีการประกาศชื่อ LCD File ที่อุปกรณ์ตัวนั้นมีอยู่ และเมื่ออุปกรณ์ที่ถูกเรียกมานั้นยังไม่เป็นหน่วยที่เล็กที่สุด คือตัวมันยังมีการเรียกอุปกรณ์ตัวอื่นอีก ผู้ใช้จะประกาศชื่อ Library นั้นๆ ไปเรื่อยจนสุดท้ายไม่มีการเรียก Library ที่เล็กลงไปอีก

ผลลัพธ์ที่ได้จากโปรแกรมคือ *.C เป็น Source File ในรูปแบบของภาษาซี และไฟล์ *.h เป็น Library ที่ *.C สามารถเรียกใช้ได้ในการ Compile

โอกาสในการเกิด Error อาจมีสาเหตุมาจาก ประกาศชื่อ Library ที่มีการเรียกใช้ไม่ครบโปรแกรมจะฟ้องมาในรูปของ Undefine อุปกรณ์ตัวที่ไม่ได้เรียก Library นั้น

4.3.2 แปลง SIM File โดยใช้โปรแกรม LCS.EXE มีรูปแบบคำสั่งดังนี้

LCS (*.SIM)

ในที่นี้ *.SIM หมายถึง SIM File ที่เขียนไว้แล้ว โดยผลลัพธ์ที่ได้คือ EXE. C เป็น Source file ในรูปของภาษาซี

โอกาสเกิด error อาจเป็นผลมาจากการประกาศ Input และ Output ไม่ตรงกับ Input Output ที่ประกาศใน LCD File

4.3.3 นำ Source File ทั้ง 2 มาทำการ Compile พร้อมกันโดย LCSS.LIB เป็น Library File ที่ใช้ในการ Complier มีรูปแบบคำสั่งดังนี้

TCC (*.LCD) EXE LCSS.LIB

ขั้นตอนในการ Compile โปรแกรม จะมีการเรียก Include File (*.h) เพราะฉะนั้น disk ที่ทำการโปรแกรม Subdirectory ชื่อ \h อยู่ Root Directory และมี stdio.h กับ Stdarg.h อยู่บน Root Directory

หากการ Compile เป็นไปอย่างเรียบร้อย ผลที่ได้จะเป็น Execute File โดยมีชื่อเดียวกับชื่อที่ตั้งให้กับ LCD File

4.3.4 ทำการรัน Execute File โดยให้ *.DAT เป็นข้อมูลของ Input ใช้คำสั่งดังนี้

(*.EXE) (*.DAT)

ได้ผลลัพธ์เป็น *.out โอกาสของการเกิด error คือ จำนวน Input ที่ป้อนให้ไม่เท่ากับจำนวน Input ที่ประกาศใน TESTIN ของ SIM File

4.3.5 รันโปรแกรม LCG.EXE โดยมี *.out เป็น Input File มีรูปแบบดังนี้

LCG (*.out)

ผลที่ได้จะออกมาในรูปของ Timing Diagram แสดงผลการ Simulate วงจรโดยแนวตั้งแสดงสัญญาณ output แต่ละเส้นและแนวตั้งเป็น Time แสดง cycle ของการทำงาน

บทที่ 5 ผลทดลองการทำงาน

หลังจากที่ได้ศึกษาทางทฤษฎีของ Digital Logic Simulator มาในบทนี้ จะกล่าวถึงการนำ software ตัวนี้มาใช้งานจริง โดยมีการทดลอง Simulate วงจรต่างๆ แบ่งเป็นข้อๆ ตามลักษณะของวงจรได้ดังนี้

5.1 ทดลองวงจร GATE พื้นฐาน

ในหัวข้อนี้ได้มีการทดลอง NAND GATE 2 Input เขียน LCD File โดยให้ชื่อว่า ND2.LCD และ เขียน ND2.SIM , ND2.DAT เพื่อทำการ Simulate แต่ในทางปฏิบัติจะทำการรวมLCD File ของอุปกรณ์พื้นฐานเหล่านี้ไว้ใน GATE.LCD ทั้งนี้เพื่อความสะดวกในการเรียกมาใช้งาน

```
>TYPE ND2.LCD      ( ENTER )

NAME      ND2;
INPUT     A, B;
OUTPUT    Y;
DEFINE
{
     Y = ~(A & B);
}
END;

>TYPE ND2.SIM      ( ENTER )

TEST      ND2;
TESTIN    A,B;
TESTOUT   A,B,Y;
```

```
END;

>TYPE ND2.DAT      ( ENTER )

PATTERN
{
      :0  0
      :0  1
      :1  0
      :1  1
      :0  0
      :1  1
      :1  0
      :0  0
      :1  1
}
END;
```

หลังจากได้ TEXT File 3 ไฟล์เรียบร้อยแล้ว ใช้คำสั่งเหล่านี้ในการ Simulate

```
>LCT ND2                ( ENTER )
>LCS ND2                ( ENTER )
>TCC ND2 EXE LCSS.LIB ( ENTER )
>ND2 ND2                ( ENTER )
>LCG ND2                ( ENTER )
```

ผลสุดท้ายเราจะได้ Timing Diagram ที่ Simulate อุปกรณ์ NAND GATE โดยมี Input ตาม pattern ที่กำหนดไว้ในไฟล์ ND2.DAT Timing Diagram แสดงไว้ในรูปที่ 5.1 สามารถพิสูจน์ให้ได้ว่า software สามารถทำการ Simulate ผลออกมาได้อย่างถูกต้อง

5.2 ทดลองวงจร Flip_Flop ตัวเดียว

การทดลองต่อวงจร Flip-Flop ในบทนี้ เราเลือกใช้ JK Flip-Flop แบบที่มี Clock แสดงวงจรได้ด้วยรูป 5.2a โดยเริ่มจากการเขียน LCD File ได้ดังนี้

```
>TYPE JKFF.LCD          ( ENTER )

NAME    JKFF;
INPUT   J, K, CK;
OUTPUT  Q, Q_;
DEFINE
{
     EXT   ND2,NOT;
     LOC   G1,G2,G3,G4,G5,G6,G7,G8,G9;

     G1 = ND3 (J,Q_,CK);
     G2 = NOT (CK);
     G3 = ND3 (K,Q,CK);
     G4 = ND2 (G1,G5);
     G5 = ND2 (G3,G4);
     G6 = ND2 (G4,G2);
     G7 = ND2 (G5,G2);
     G8 = ND2 (G6,G9);
     G9 = ND2 (G7,G8);
```

```
    Q  = G8;
    Q_ = G9;
}
END;
```

JK Flip-Flop ตัวนี้สร้างโดยให้มี Clock และ วงจรจะทำงานในช่วงขาลงของ Clock ต่อมาเขียน SIM File ได้ดังนี้

```
>TYPE JKFF.SIM          ( ENTER )

TEST     JKFF;
TESTIN   J, K, CK;
TESTOUT  J, K, CK, Q, Q_;
END;
```

ใน DAT File ได้มีการทดลองกำหนดข้อมูล Input ไว้ดังนี้

```
>TYPE    JKFF.DAT       ( ENTER )

PATTERN
{
  /* J K CK */
    :0 1 1
    :1 0 0
    :1 1 1
    :0 1 0
    :0 1 1
    :1 0 0
```

```
        :0 0 1
        :1 1 0
        :0 1 1
        :0 1 0
        :1 1 1
        :0 0 0
        :0 0 1
        :1 0 0
        :1 1 1
        :1 1 0
        :0 0 1
        :1 0 0
        :0 0 1
        :0 1 1
        :0 1 0
        :1 1 1
        :0 0 0
        :0 0 1
        :0 0 0
}
END;
```

ต่อมาใช้คำสั่งเหล่านี้เพื่อ Simulate

```
>LCT JKFF GATE          ( ENTER )
>LCS JKFF               ( ENTER )
>TCC JKFF EXE           ( ENTER )
>JKFF JKFF              ( ENTER )
```

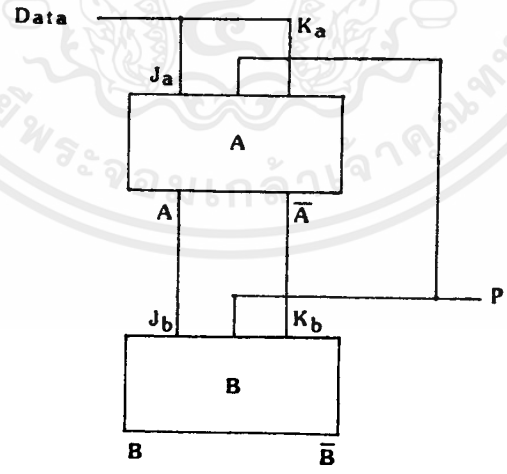
>LOG JKFF (ENTER)

ผลที่ได้ในที่สุดเป็นไปในรูป Timng Diagram แสดงการทำงานของ JK Flip-Flop โดย Input ที่ป้อนให้มันถูกกำหนด pattern ไว้ใน JKFF.DAT จากรูปที่ 5.2b จะเห็นว่า Timing Diagram มี Clock กำหนด Cycle ในการทำงานของอุปกรณ์ด้วย

5.3 ทดลองวงจร Flip-Flop หลายตัว

การนำ Flip-Flop หลายตัวมาต่อกันโดยมีรูปแบบที่ต่างกัน ทำให้เกิดวงจรที่มีหน้าที่การทำงานต่างกันได้หลายลักษณะดังนี้

5.3.1 วงจรดังรูปที่ 5.3.1a



รูปที่ 5.3.1a แสดงวงจรที่มี JKFF 2 ตัว ใช้ในการส่งข้อมูล

เป็นวงจรที่มีการส่งผ่านข้อมูลจาก JKFF A ไปยัง JKFF B โดยมี P แทน Clock กำหนดจังหวะในการส่ง เขียนแทนด้วยสัญลักษณ์ได้ดังนี้

P: A -> B

LCD File สามารถเขียนได้ดังนี้

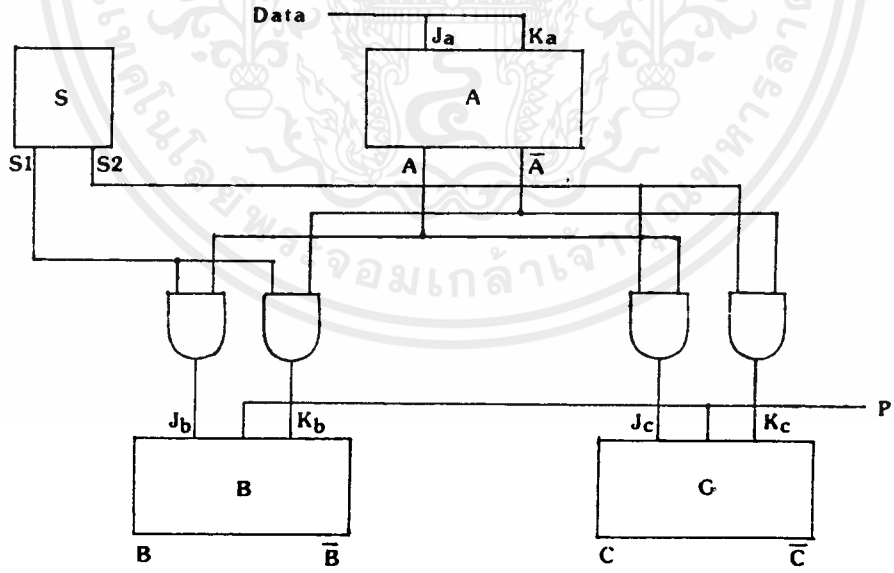
```
>TYPE   5_3_1.LCD        ( ENTER )

NAME    FIG5_3_1;
INPUT   D1,D2,P;
OUTPUT  A,A_,B,B_;

DEFINE
{
    EXT     JKFF;
    LOC     U1,U2;

    U1 = JKFF (D1,D2,P);
    U2 = JKFF (U1.Q,U1.Q_,P);
    A  = U1.Q;
    A_ = U1.Q_;
    B  = U2.Q;
    B_ = U2.Q_;
}
END;
```

ในการ Compile ด้วย LCT.EXE ต้องมีการเรียกด้วยคำสั่งดังนี้

```
>LCT 5_3_1 JKFF GATE      ( ENTER )
>LCS 5_3_1                ( ENTER )
>TCC 5_3_1 EXE LCSS.LIB   ( ENTER )
>5_3_1 5_3_1              ( ENTER )
>LCG 5_3_1                ( ENTER )
```

ผลที่ได้จะเป็นในรูปของ Timing Diagram แสดงการส่งข้อมูลจาก JKFF A ไปยัง JKFF B ดังแสดงไว้ในรูปที่ 5.3.1b

5.3.2 วงจรดังรูปที่ 5.3.2a



รูปที่ 5.3.2a แสดงวงจรที่มี JKFF 3 ตัว ใช้ในการส่งข้อมูล

เป็นวงจรที่ประกอบด้วย JKFF 3 ตัว คือ JKFF A, JKFF B และ JKFF C โดยที่ JKFF A สามารถเลือกได้ว่าจะส่งข้อมูลไปยัง JKFF B หรือ JKFF C โดยมี S1, S2 เป็นตัวเลือก เขียนแทนด้วยสัญลักษณ์ได้ดังนี้

P: if S1 = 1, S2 = 0 then A -> B,
if S1 = 0, S2 = 1 then A -> C

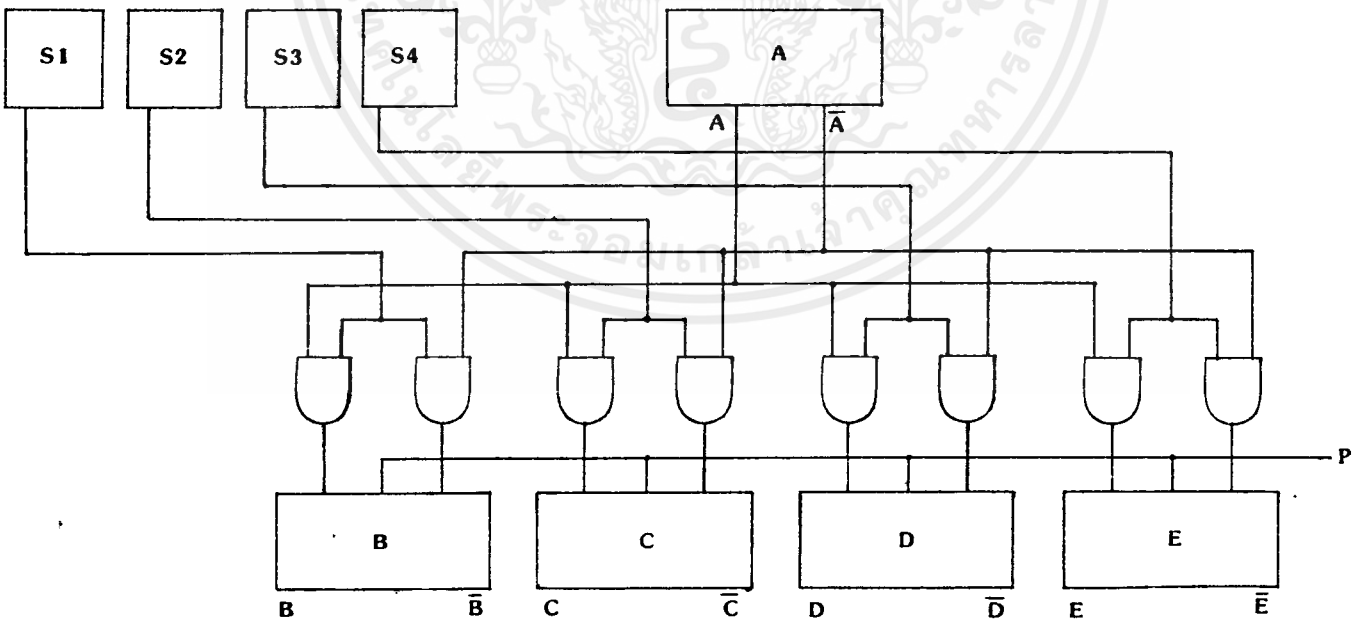
LCD File สามารถเขียนได้ดังนี้

```
>TYPE     5_3_2.LCD        ( ENTER )

NAME      FIG5_3_2;
INPUT     S1,S2,D1,D2,P;
OUTPUT    A,A_,B,B_,C,C_;
DEFINE
{
     EXT       AND2,JKFF;
     LOC       U1,U2,U3,G1,G2,G3,G4;

     U1 = JKFF (D1,D2,P);
     U2 = JKFF (G1,G2,P);
     U3 = JKFF (G3,G4,P);
     G1 = AND2 (S1,A);
     G2 = AND2 (S1,A_);
     G3 = AND2 (S2,A);
     G4 = AND2 (S2,A_);
     A  = U1.Q;
     A_ = U1.Q_;
```

```
        B  = U2.Q;
        B_ = U2.Q_;
        C  = U3.Q;
        C_ = U3.Q_;
}
END;
```

ขั้นตอนในการ Compile ใช้คำสั่งเดียวกันกับหัวข้อที่ 5.3.1 ได้ Timing Diagram แสดงผลการทำงานของวงจรตามรูป 5.3.2b จะเห็นว่าเมื่อ S1 เป็น "1" ข้อมูลจาก FF A จะถูกส่งไปยัง FF B ในทางกลับกัน ถ้า S2 เป็น "1" ข้อมูลจาก FF A จะถูกส่งไปยัง FF C

5.3.3 วงจรดังรูปที่ 5.3.3a



รูปที่ 5.3.3a แสดงวงจรที่มี JKFF 5 ตัว ใช้ในการส่งข้อมูล

เป็นวงจรที่ประกอบด้วย JKFF 5 ตัว คือ JKFF A, B, C, D และ E โดยมี JKFF A เป็นตัวส่งข้อมูล และ JKFF B, C, D, E เป็นตัวรับข้อมูล S1-S4 ทำหน้าที่เลือกว่า ข้อมูลจาก FF A จะถูกส่งไปยัง FF ตัวไหน เขียนแทนด้วยสัญลักษณ์

```
P: if S1 = 1 then A -> B,
   if S2 = 1 then A -> C,
   if S3 = 1 then A -> D,
   if S4 = 1 then A -> E
```

LCD File สามารถเขียนได้ดังนี้

```
>TYPE   5_3_3.LCD        ( ENTER )

NAME    FIG5_3_3;
INPUT   S1,S2,S3,S4,D1,D2,P;
OUTPUT  A,A_,B,B_,C,C_,D,D_,E,E_;
DEFINE
{
     EXT      AND2,JKFF;
     LOC      U1,U2,U3,U4,U5,G1,G2,G3,G4,G5,G6,G7,G8;

     U1 = JKFF (D1,D2,P);
     U2 = JKFF (G1,G2,P);
     U3 = JKFF (G3,G4,P);
     U4 = JKFF (G5,G6,P);
     U5 = JKFF (G7,G8,P);
     G1 = AND2 (S1,A);
     G2 = AND2 (S1,A_);
```

```
        G3 = AND2 (S2,A);
        G4 = AND2 (S2,A_);
        G5 = AND2 (S3,A);
        G6 = AND2 (S3,A_);
        G7 = AND2 (S4,A);
        G8 = AND2 (S4,A_);
        A  = U1.Q;
        A_ = U1.Q_;
        B  = U2.Q;
        B_ = U2.Q_;
        C  = U3.Q;
        C_ = U3.Q_;
        D  = U4.Q;
        D_ = U4.Q_;
        E  = U5.Q;
        E_ = U5.Q_;
    }
END;
```

ขั้นตอนในการ Compile ใช้คำสั่งในลักษณะเดียวกันกับหัวข้อ 5.3.1 ได้ Timing Diagram ดังแสดงในรูปที่ 5.3.3b จะเห็นว่าเมื่อ

S1 = 1 ข้อมูลจาก FF A จะถูกส่งไปยัง FF B

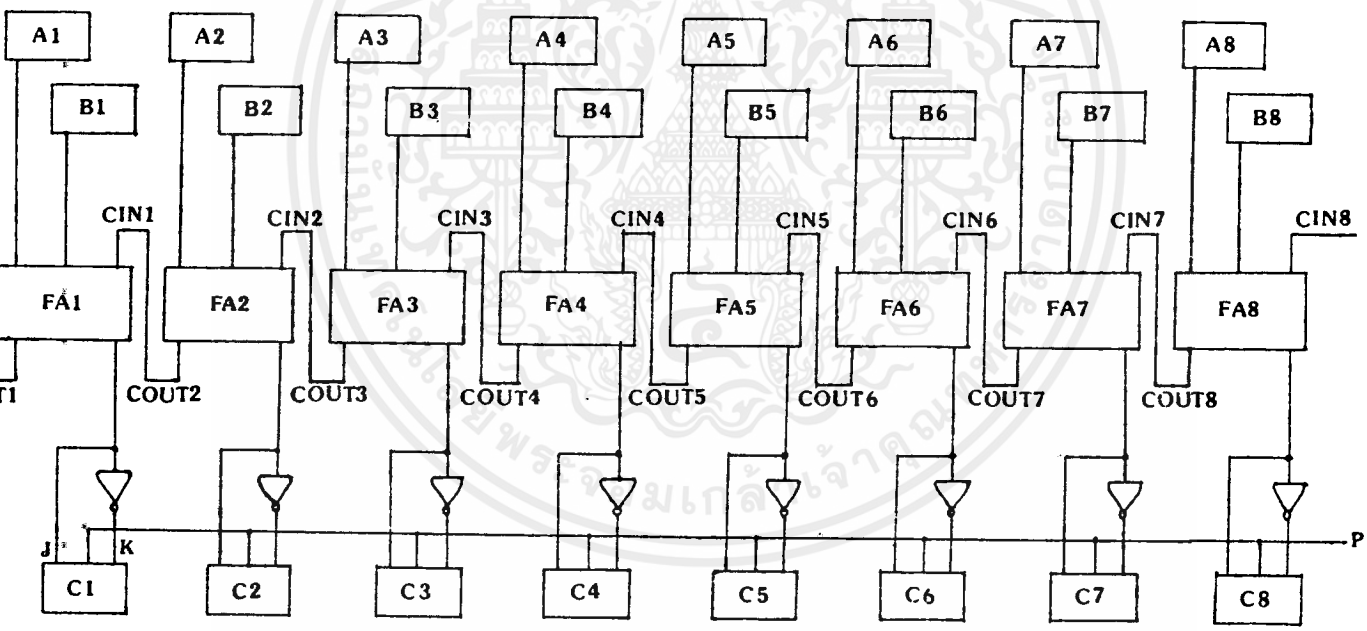
S2 = 1 ข้อมูลจาก FF A จะถูกส่งไปยัง FF C

S3 = 1 ข้อมูลจาก FF A จะถูกส่งไปยัง FF D

S4 = 1 ข้อมูลจาก FF A จะถูกส่งไปยัง FF E

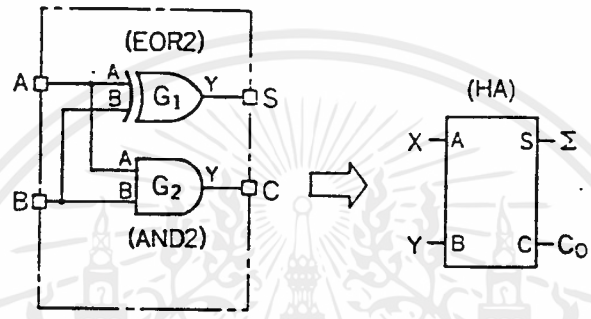
5.4 ทดลองวงจรประเภท Adder

จากวงจรจะเห็นว่าเป็นวงจรบวกเลข 2 byte เข้าด้วยกัน โดยแต่ละ byte มีจำนวน 8 bit ประกอบไปด้วย Full Adder 8 ตัว และมี JKFF อีก 8 ตัวทำหน้าที่เก็บผลลัพธ์ที่ได้จากการบวกแต่ละครั้ง ดังแสดงในรูปที่ 5.4



รูปที่ 5.4 แสดงวงจรบวกเลข 8 bit

ขั้นแรกเราต้องทำการเขียน Library ของ Full Adder และ Half Adder โดยเริ่มจาก Half Adder ซึ่งเป็นวงจรบวกเลขแบบไม่มีตัวทดเข้า ประกอบด้วย AND Gate และ EOR Gate ดังรูปที่ 5.4.1



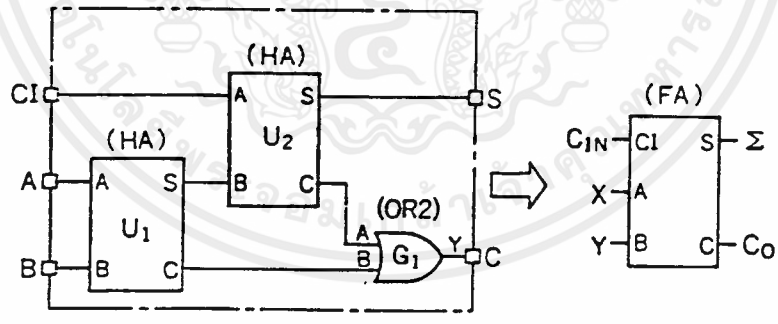
รูปที่ 5.4.1 แสดงวงจร Half Adder พร้อมวงจรสมมูลย์

จากรูปสามารถสร้าง LCD File โดยให้ชื่อว่า HA.LCD ได้ดังนี้

```
>TYPE    HA.LCD          ( ENTER )

NAME     HA;
INPUT    A, B;
OUTPUT   C, S;
DEFINE
{
     EXT EOR2, AND2;
     LOC G1, G2;
```

```
        G1   = EOR2 (A, B);
        G2   = AND2 (A, B);
        S    = G1;
        C    = G2;
}
END;
```

ต่อมาสร้าง Library ของ Full Adder ซึ่งเป็นวงจรบวกเลขแบบมีตัวทดเข้า ดังแสดงในรูปที่ 5.4.2



รูปที่ 5.4.2 แสดงวงจร Full Adder พร้อมวงจรสมมูลย์

จากรูปสามารถสร้าง LCD File โดยใช้ชื่อว่า FA.LCD ได้ดังนี้

```
>TYPE    FA.LCD          ( ENTER )

NAME     FA;
INPUT    C1, A, B;
OUTPUT   C, S;
DEFINE
{
     EXT EOR2, HA;
     LOC G1, U1, U2;

     G1   = OR2 (U1.C, U2.C);
     U1   = HA (A, B);
     U2   = HA (C1, U1.S);
     S    = U2.S;
     C    = G1;
}
END;
```

เมื่อได้ Library พื้นฐานที่ใช้ในวงจรแล้ว ทำการสร้าง LCD File ของวงจรบวกเลข 8 bit โดยให้ชื่อว่า 5_4.LCD ได้ดังนี้

```
>TYPE    5_4.LCD         ( ENTER )

NAME     FIG5_4;
INPUT    A1,A2,A3,A4,A5,A6,A7,A8,B1,B2,B3,B4,B5,B6,B7,B8,CIN8,P;
```

```
OUTPUT   C1,C2,C3,C4,C5,C6,C7,C8,D1,D2,D3,D4,D5,D6,D7,D8,COUT1;

DEFINE
{
     EXT       JKFF,FA,NOT;
     LOC       FA1,FA2,FA3,FA4,FA5,FA6,FA7,FA8,JKFF1,JKFF2,JKFF3,JKFF4,JKF

     FA8 = FA ( CIN8,A8,B8);
     FA7 = FA ( FA8.C,A7,B7);
     FA6 = FA ( FA7.C,A6,B6);
     FA5 = FA ( FA6.C,A5,B5);
     FA4 = FA ( FA5.C,A4,B4);
     FA3 = FA ( FA4.C,A3,B3);
     FA2 = FA ( FA3.C,A2,B2);
     FA1 = FA ( FA2.C,A1,B1);

     JKFF1 = JKFF ( FA1.S,G1,P);
     JKFF2 = JKFF ( FA2.S,G2,P);
     JKFF3 = JKFF ( FA3.S,G3,P);
     JKFF4 = JKFF ( FA4.S,G4,P);
     JKFF5 = JKFF ( FA5.S,G5,P);
     JKFF6 = JKFF ( FA6.S,G6,P);
     JKFF7 = JKFF ( FA7.S,G7,P);
     JKFF8 = JKFF ( FA8.S,G8,P);

     G1 = NOT (FA1.S);
     G2 = NOT (FA2.S);
     G3 = NOT (FA3.S);
```

```
        G4 = NOT (FA4.S);
        G5 = NOT (FA5.S);
        G6 = NOT (FA6.S);
        G7 = NOT (FA7.S);
        G8 = NOT (FA8.S);

        C1 = JKFF1.Q;
        C2 = JKFF2.Q;
        C3 = JKFF3.Q;
        C4 = JKFF4.Q;
        C5 = JKFF5.Q;
        C6 = JKFF6.Q;
        C7 = JKFF7.Q;
        C8 = JKFF8.Q;

        COUT1 = FA1.C;

        D1 = FA1.S;
        D2 = FA2.S;
        D3 = FA3.S;
        D4 = FA4.S;
        D5 = FA5.S;
        D6 = FA6.S;
        D7 = FA7.S;
        D8 = FA8.S;
    }
    END;
```

ทำการแปลง LCD File โดยใช้ LCT.EXE ด้วยคำสั่งดังนี้

```
>LCT 5_4 JKFF FA HA GATE      ( ENTER )
```

ทำตามขั้นตอนต่างๆในการแปลง จนในที่สุดก็จะได้ EXECUTE File ชื่อ 5_4.EXE จากนั้นสร้าง 5_4.DATเพื่อกำหนด Input ที่เป็นเลขฐานสอง จำนวน 8 bit 2 byte ที่ต้องการบวกกัน ข้อมูลใน 5_4.DAT แสดงได้ดังนี้

```
>TYPE   5_4.DAT        ( ENTER )
PATTERN
{
/*    A1A2A3A4 A5A6A7A8    B1B2B3B4 B5B6B7B8  C P   */
     :0 0 0 0  0 0 0 0     0 0 0 0  0 0 0 0   0 1
     :0 0 0 0  0 0 0 0     0 0 0 0  0 0 0 0   0 0

     :0 0 0 0  0 0 0 0     0 0 0 0  0 0 0 0   0 1
     :0 0 0 0  0 0 0 0     0 0 0 0  0 0 0 0   0 0

     :0 0 0 0  0 0 0 1     0 0 0 0  0 0 0 1   0 1
     :0 0 0 0  0 0 0 1     0 0 0 0  0 0 0 1   0 0

     :0 0 0 0  0 0 1 0     0 0 0 0  0 0 1 0   0 1
     :0 0 0 0  0 0 1 0     0 0 0 0  0 0 1 0   0 0

     :0 0 0 0  0 0 1 1     0 0 0 0  0 0 1 1   0 1
     :0 0 0 0  0 0 1 1     0 0 0 0  0 0 1 1   0 0
```

```
G5      = AND3 ( A_, B, C);
G6      = AND3 ( A_, B, C_);
G7      = AND3 ( A_, B_, C);
G8      = AND3 ( A_, B_, C_);
G9      = OR4  ( G1, G5, G6, G7);
G10     = OR4  ( G1, G2, G7, G8);
G11     = OR4  ( G4, G6, G7, G8);
G12     = OR2 ( G2, G3);
G13     = OR3 ( G2, G3, G4);
G14     = OR4 (G2, G3, G4, G7);
G15     = OR2 (G1, G7);
G16     = OR4 (G3, G4, G6, G7);
G17     = OR2 (G1, G6);
G18     = OR2 (G2, G5);
G19     = OR4 (G2, G3, G4, G5);
G20     = OR2 (G3, G5);
G21     = OR2 (G5, G5);
G22     = NOT ( G9);
G23     = NOT ( G10);
G24     = NOT ( G11);

a       = G12;
b       = G13;
c       = G14;
d       = G15;
e       = G16;
f       = G17;
g       = G18;
```

:0 0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 1

:0 0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0

:0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 1

:0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0

:0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 1

:0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0

:0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1

:0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0

:0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1

:0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0

:0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 1

:0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0

:0 0 0 1 0 0 1 1 0 0 0 1 0 0 1 1 0 1

:0 0 0 1 0 0 1 1 0 0 0 1 0 0 1 1 0 0

:0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 1

:0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0

:0 0 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1

:0 0 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 0

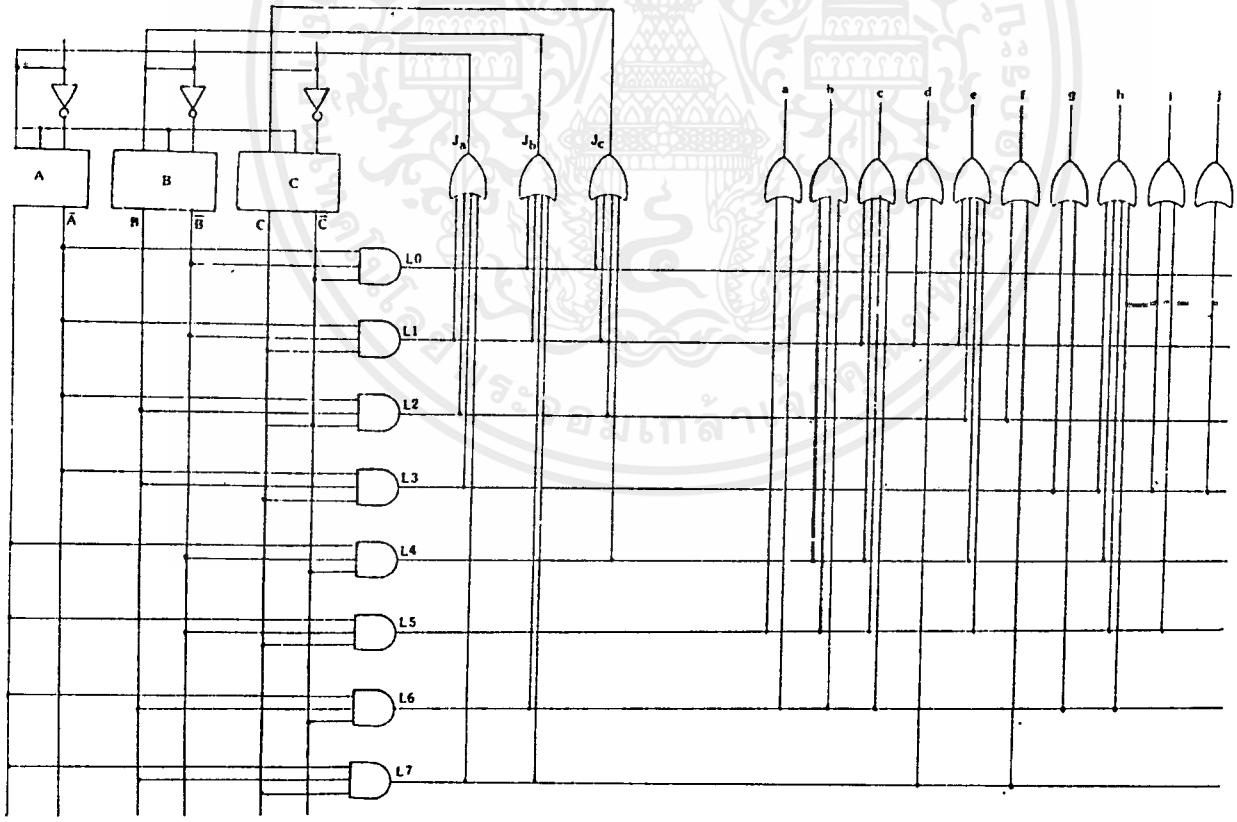
```
:0 0 0 1  0 1 1 0    0 0 0 1  0 1 1 0  0 1
:0 0 0 1  0 1 1 0    0 0 0 1  0 1 1 0  0 0

:0 0 0 1  0 1 1 1    0 0 0 1  0 1 1 1  0 1
:0 0 0 1  0 1 1 1    0 0 0 1  0 1 1 1  0 0

:0 0 0 1  1 0 0 0    0 0 0 1  1 0 0 0  0 1
:0 0 0 1  1 0 0 0    0 0 0 1  1 0 0 0  0 0

:0 0 0 1  1 0 0 1    0 0 0 1  1 0 0 1  0 1
:0 0 0 1  1 0 0 1    0 0 0 1  1 0 0 1  0 0

:0 0 0 1  1 0 1 0    0 0 0 1  1 0 1 0  0 1
:0 0 0 1  1 0 1 0    0 0 0 1  1 0 1 0  0 0

:0 0 0 1  1 0 1 1    0 0 0 1  1 0 1 1  0 1
:0 0 0 1  1 0 1 1    0 0 0 1  1 0 1 1  0 0

:0 0 0 1  1 1 0 0    0 0 0 1  1 1 0 0  0 1
:0 0 0 1  1 1 0 0    0 0 0 1  1 1 0 0  0 0

:0 0 0 1  1 1 0 1    0 0 0 1  1 1 0 1  0 1
:0 0 0 1  1 1 0 1    0 0 0 1  1 1 0 1  0 0

:0 0 0 1  1 1 1 0    0 0 0 1  1 1 1 0  0 1
:0 0 0 1  1 1 1 0    0 0 0 1  1 1 1 0  0 0
```

```
        :0 0 0 1  1 1 1 1     0 0 0 1  1 1 1 1  0 1
        :0 0 0 1  1 1 1 1     0 0 0 1  1 1 1 1  0 0
    }
END;
```

ทำการรันโปรแกรมจนในที่สุดก็จะได้ Timing Diagram แสดงการทำงานของวงจรบวกเลข8 bit ดังแสดงในรูปที่ 5.4.3 จะเห็นว่าผลที่ได้ตรงตามความเป็นจริง

5.5 ทดลองวงจรประเภท Sequential

ในหัวข้อนี้จะทำการทดลองเกี่ยวกับ Synchronous Sequential ดังรูปที่ 5.5



รูปที่ 5.5 แสดงวงจร Synchronous Sequential

สามารถเขียน LCD File ได้ดังนี้

```
>TYPE   5_5.LCD        ( ENTER )

NAME    FIG5_5;
INPUT   P;
OUTPUT  a,b,c,d,e,f,g,h,i,j, A, A_, B, B_, C, C_;

DEFINE
{
     EXT       JKFF,AND3,OR2,OR3,OR4,NOT;
     LOC       JKFF1,JKFF2,JKFF3,G1,G2,G3,G4,G5,G6,G7,G8,G9,G10,G11,G12,G1

     JKFF1   = JKFF ( G9, G22, P);
     JKFF2   = JKFF ( G10, G23, P);
     JKFF3   = JKFF ( G11, G24, P);

     A       = JKFF1.Q;
     A_      = JKFF1.Q_;
     B       = JKFF2.Q;
     B_      = JKFF2.Q_;
     C       = JKFF3.Q;
     C_      = JKFF3.Q_;

     G1      = AND3 ( A, B, C);
     G2      = AND3 ( A, B, C_);
     G3      = AND3 ( A, B_, C);
     G4      = AND3 ( A, B_, C_);
```

:0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1

:0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0

:0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1

:0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0

:0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 1

:0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0

:0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 1

:0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0

:0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1

:0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0

:0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 1

:0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0

:0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1

:0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0

:0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 1

:0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 0

:0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 1

:0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0

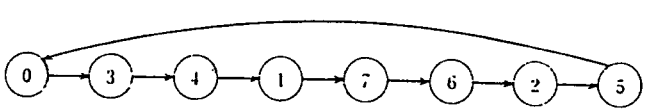
```
        h       = G19;
        i       = G20;
        j       = G21;

    }
END;
```

ทำการ compile LCD File ได้ดังนี้

```
>LCT 5_5 JKFF GATE   ( ENTER )
```

และทำตามขั้นตอนต่างๆจนได้ออกมาเป็น Timing Diagram ที่แสดงในรูปที่ 5.5.1 นำผลที่ได้มาเปรียบเทียบกับผลที่ได้ในทางทฤษฎี

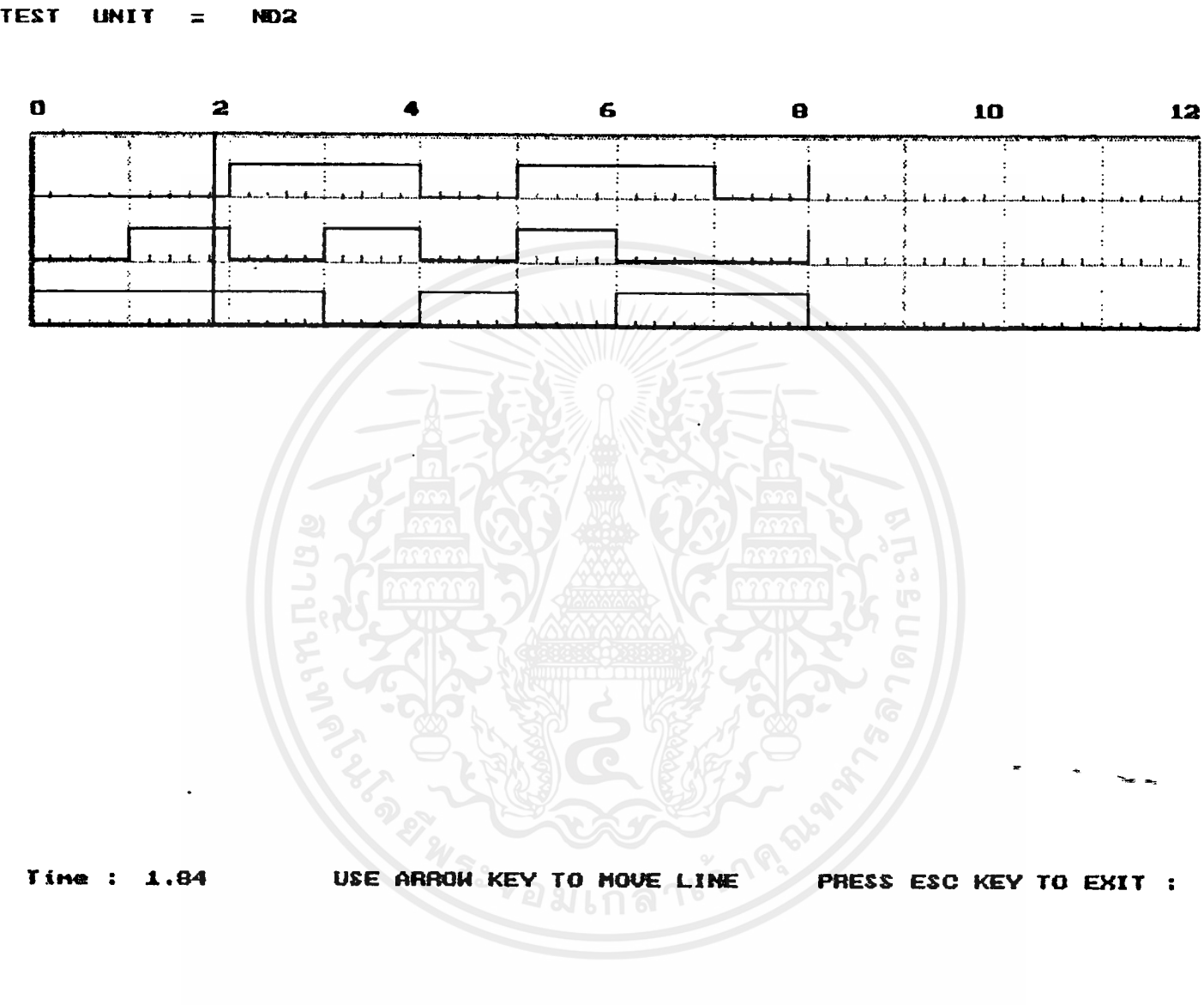
ตามทฤษฎีวงจรจะมีการทำงานตามแผนภูมิ และมีการแสดง State Table ดังต่อไปนี้



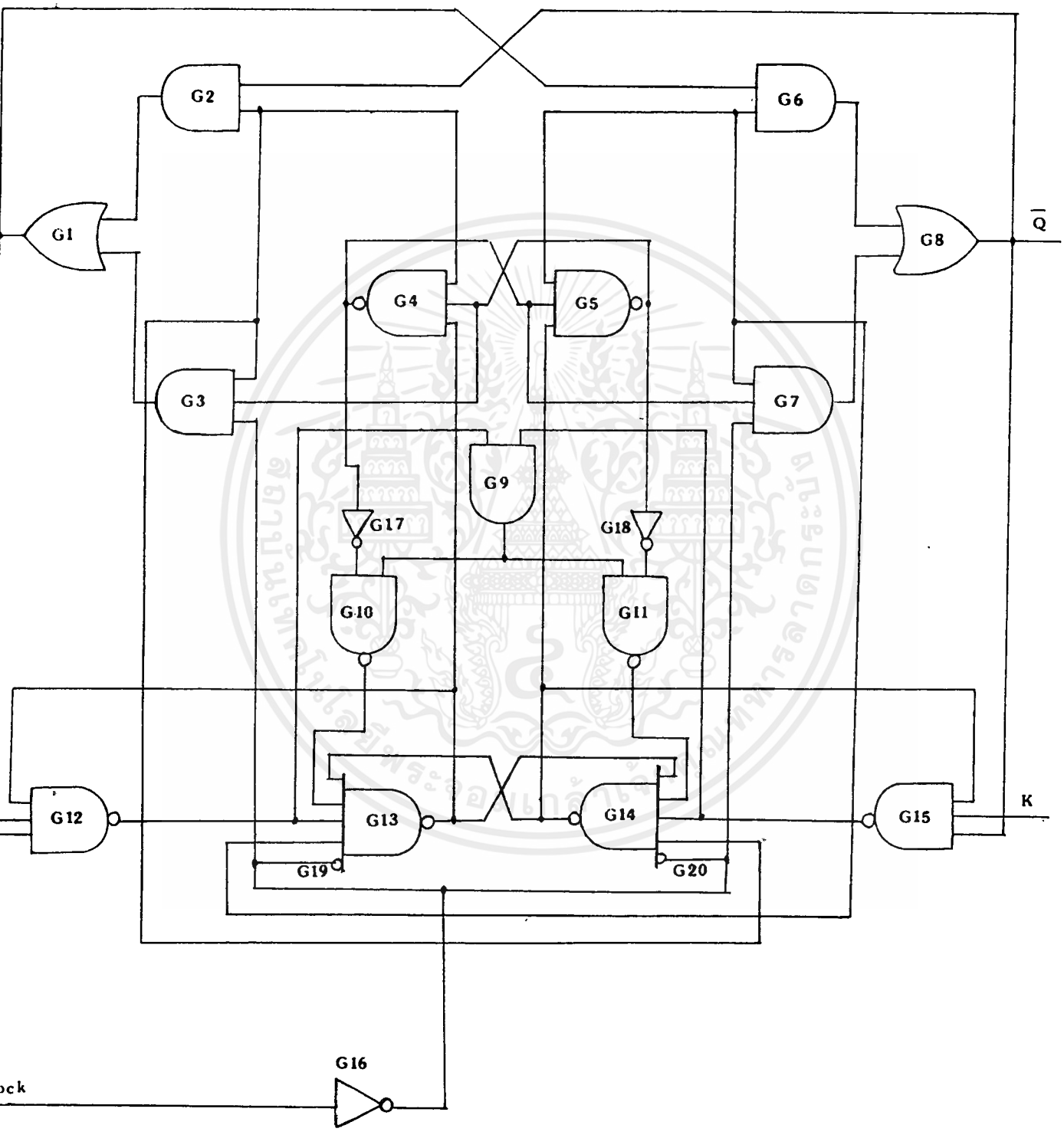
番 地	1にすべき制御情報
0	ナシ
1	c, d, e
2	e, f
3	g, h, i, j
4	b, c, e, h
5	a, b, c, e, h, i
6	a, b, c, g, h
7	d, f

เมื่อนำผลที่ได้ทางทฤษฎีมาเปรียบกับผลที่ได้จาก Timing Diagram โดยดูว่าที่เวลาหนึ่ง ขา a ถึง j ขาใดเป็น "1" บ้าง นำมาตรวจหา State ใน State Table เมื่อทราบ State แล้วไปดูต่อในแผนภูมิว่าจะเป็น State ใด ณ เวลาต่อไป และนำผลมาเทียบกับขา a ถึง j ว่าตรงกับความเป็นจริงหรือไม่ การทำงานของวงจรเช่นนี้ State ของมันจะเปลี่ยนไปตามแผนภูมิไม่มีการหยุด เป็น Cycle ไปเรื่อยๆ

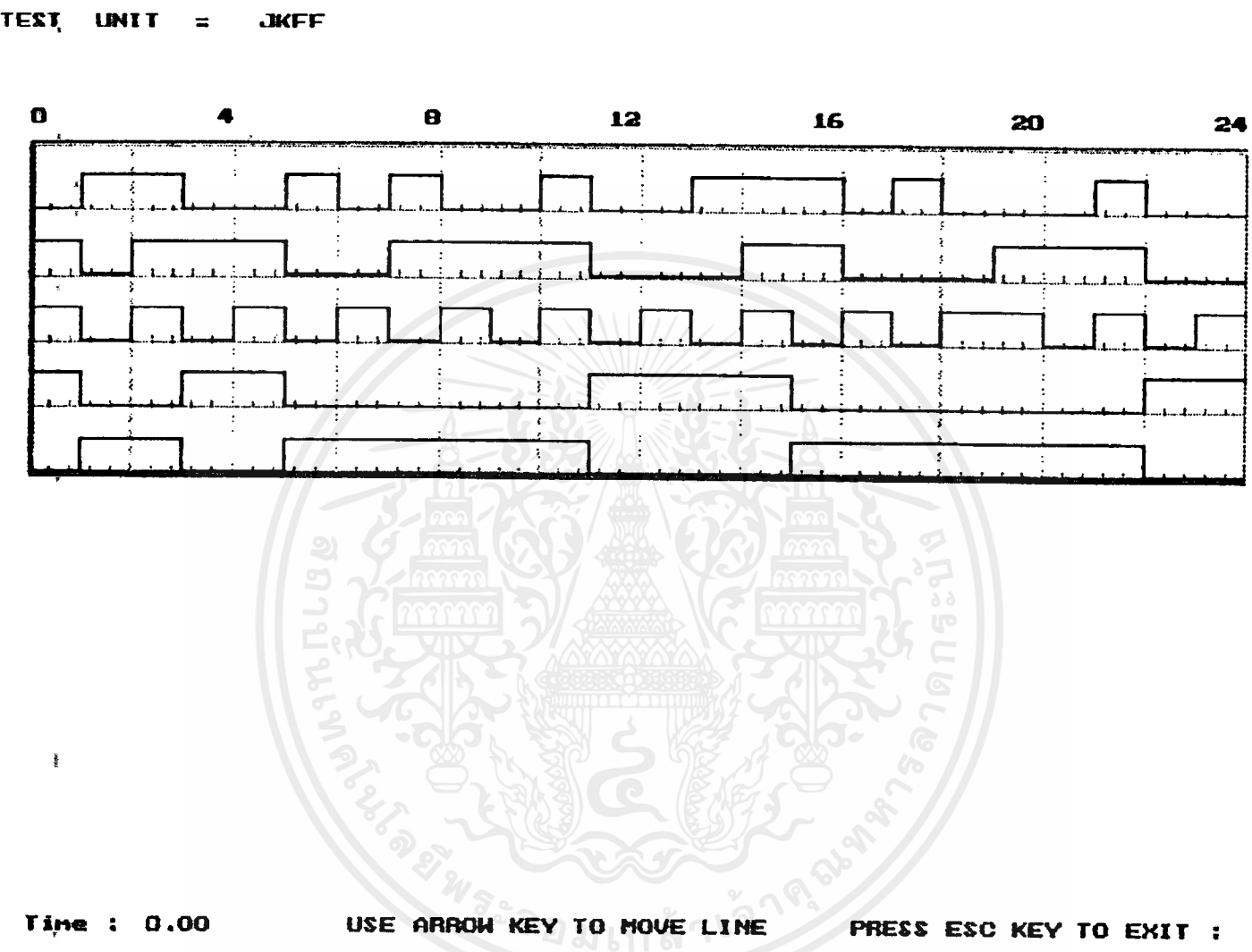
จาก Timing Diagram ของวงจรทั้งหมดที่กล่าวมาในแต่ละหัวข้อในบทนี้ จะเห็นว่าได้ผลตรงกับความเป็นจริงทุกหัวข้อ



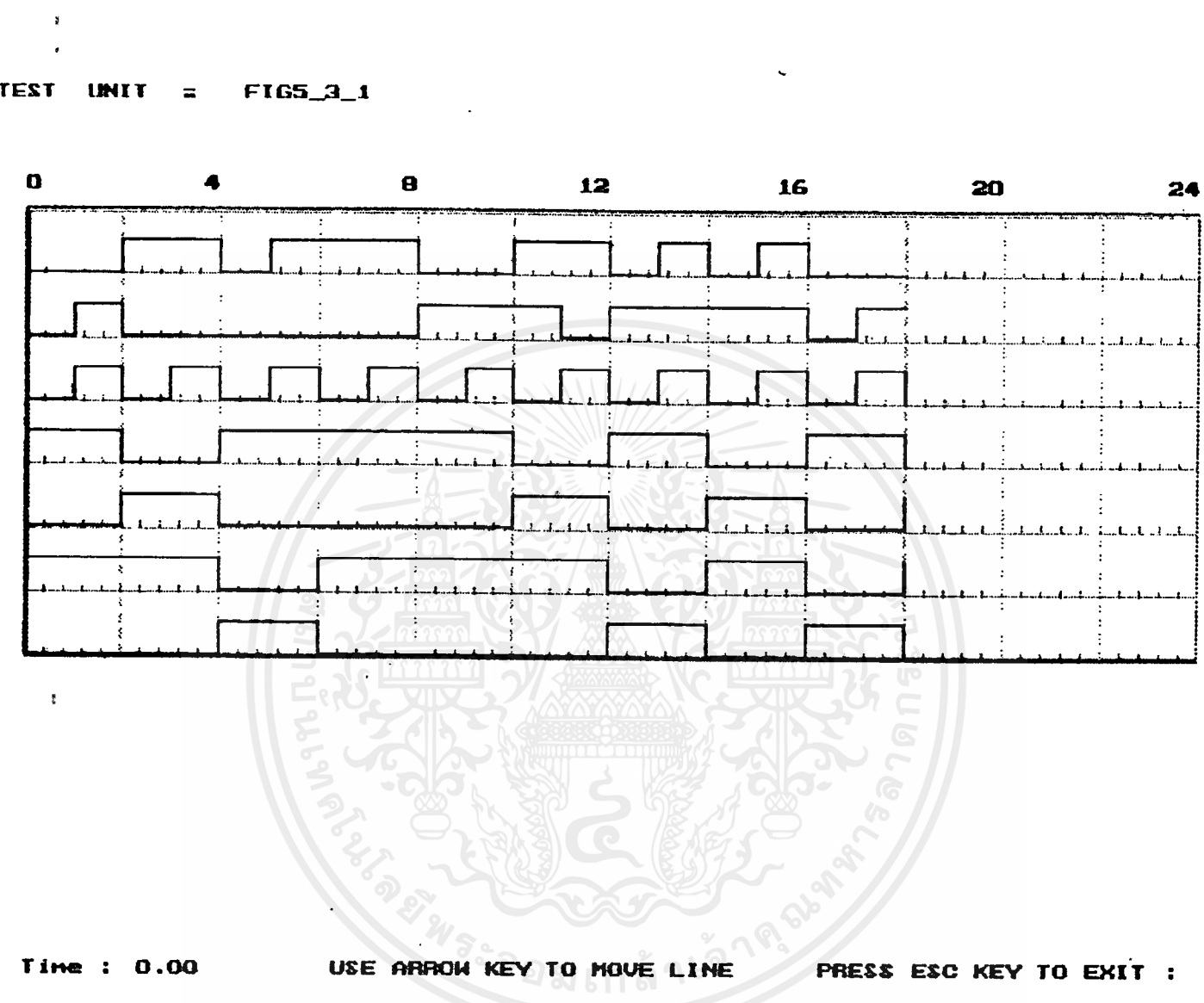
รูปที่ 5.1 แสดง ไทม์มิ่งไดอะแกรม ของ NAND GATE 2 Input



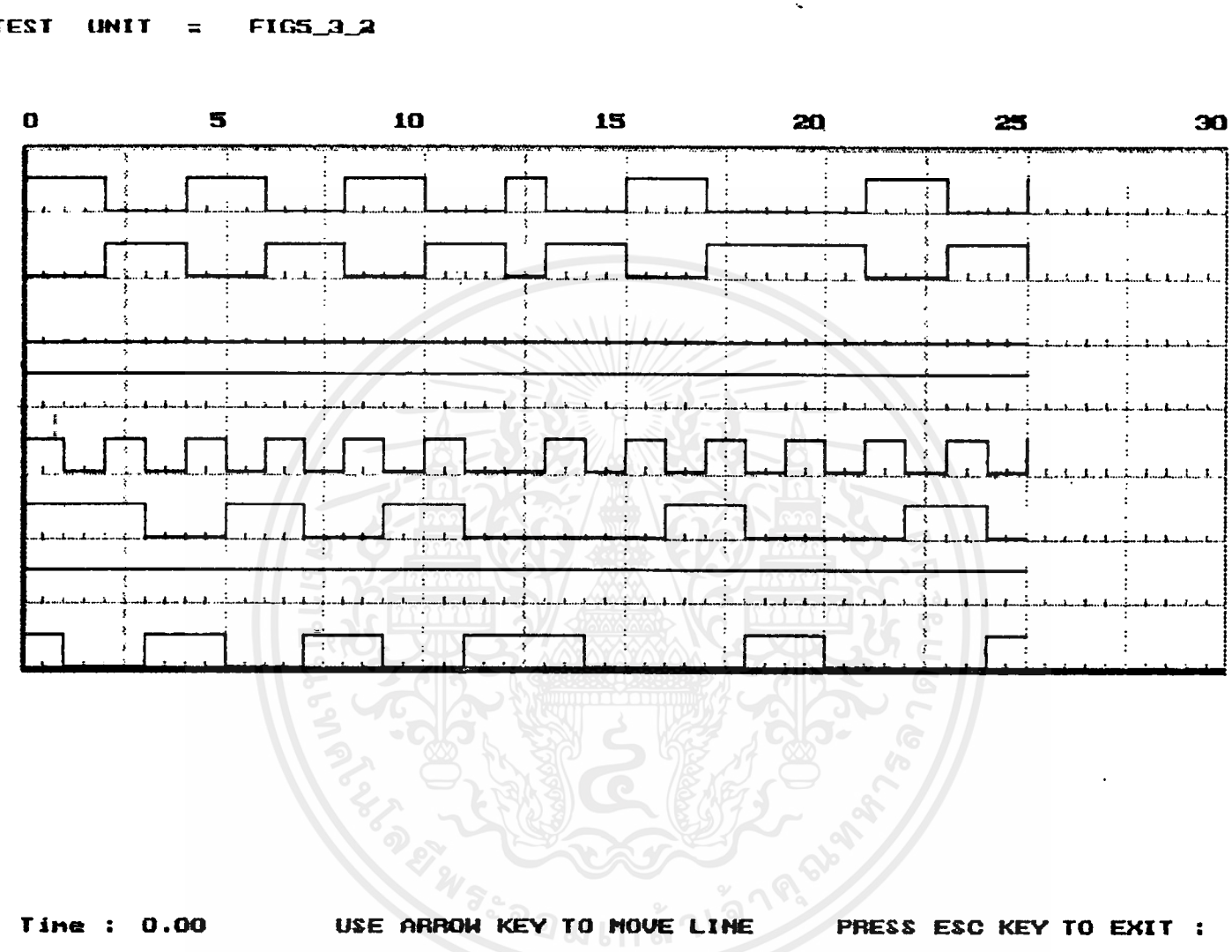
รูปที่ 5.2a แสดง JKFF แบบมี Clock



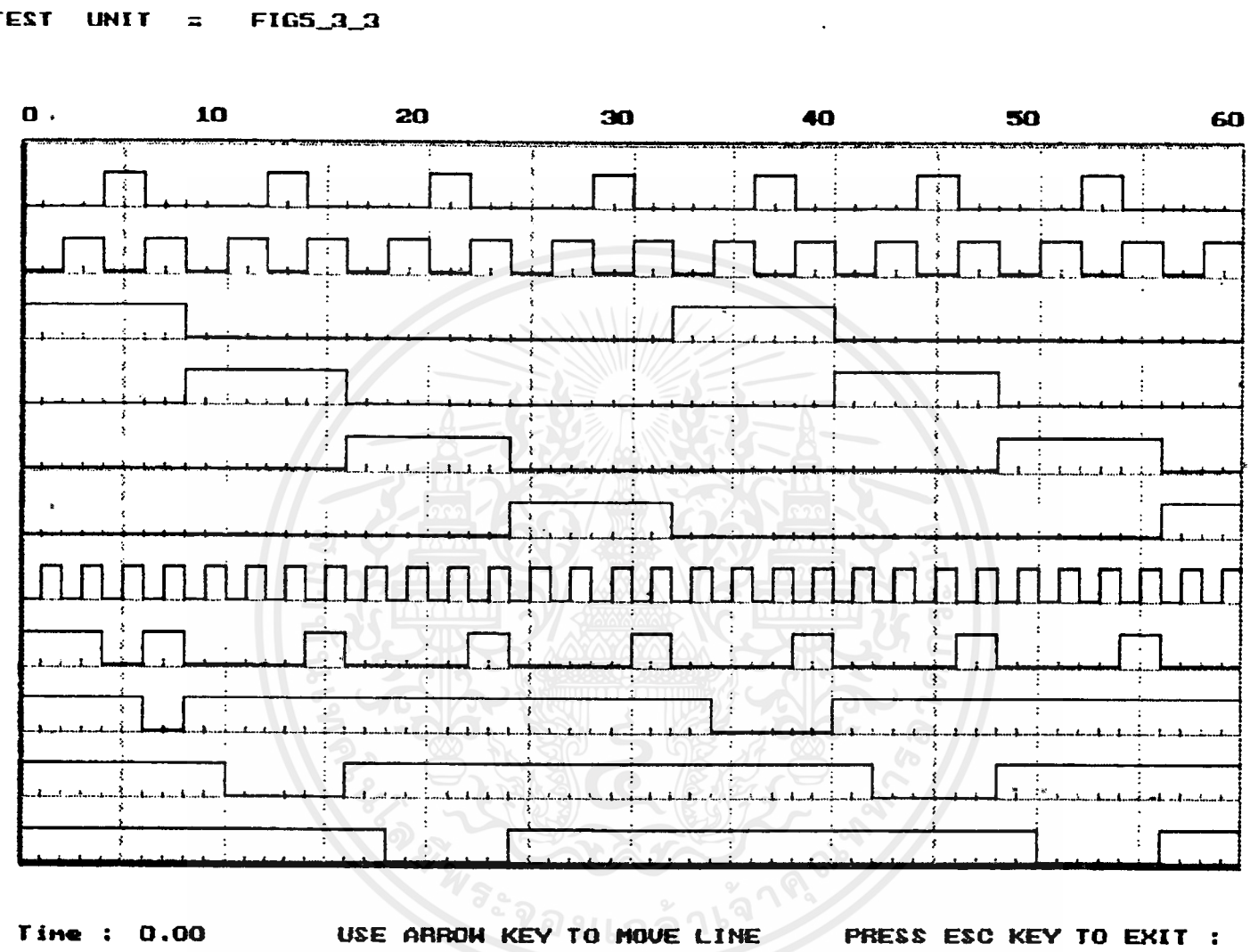
รูปที่ 5.2b แสดง Timing Diagram ของ JKFF แบบมี Clock



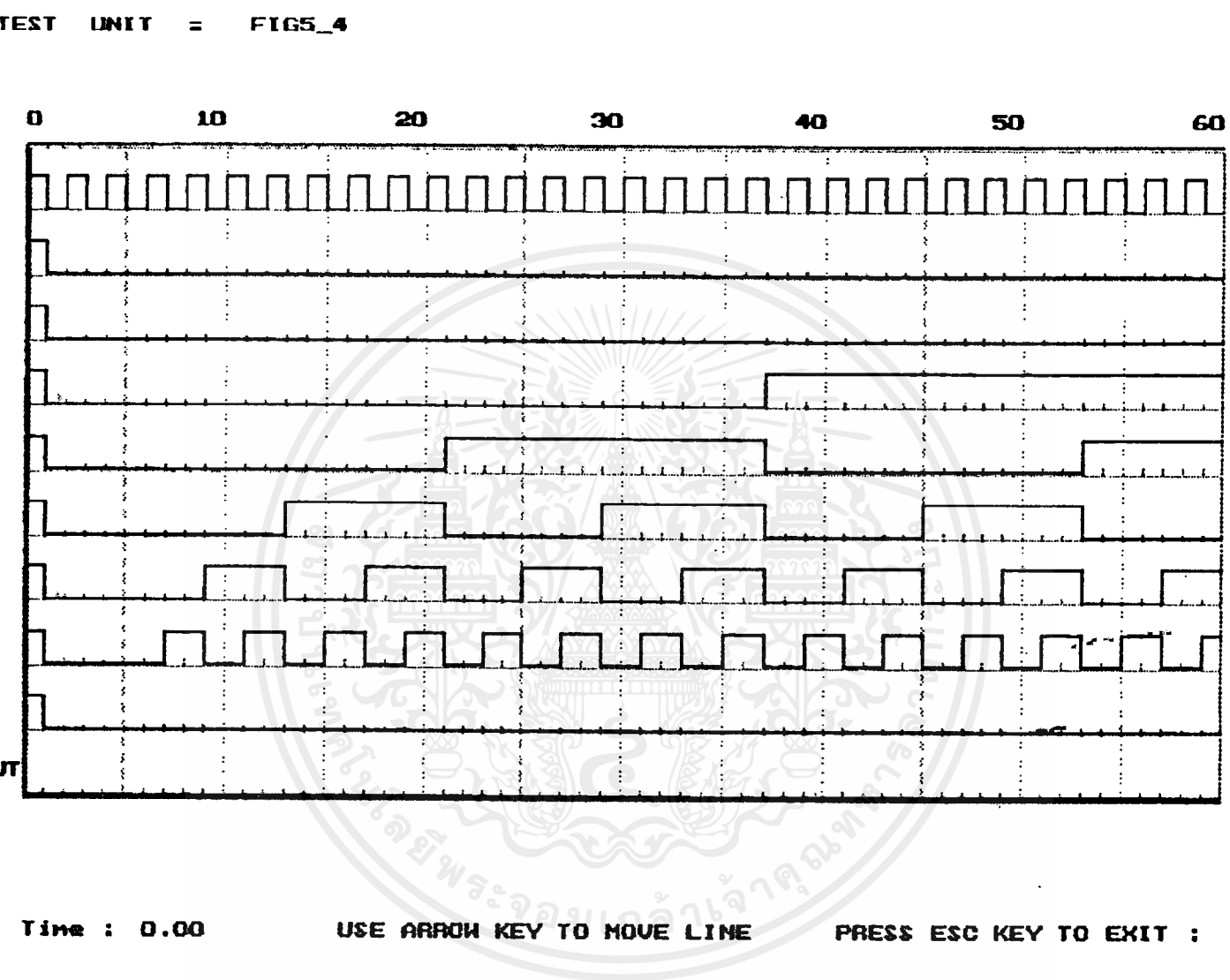
รูปที่ 5.3.1b แสดง ไทม์มิ่งไดอะแกรม ของรูปที่ 5.3.1a



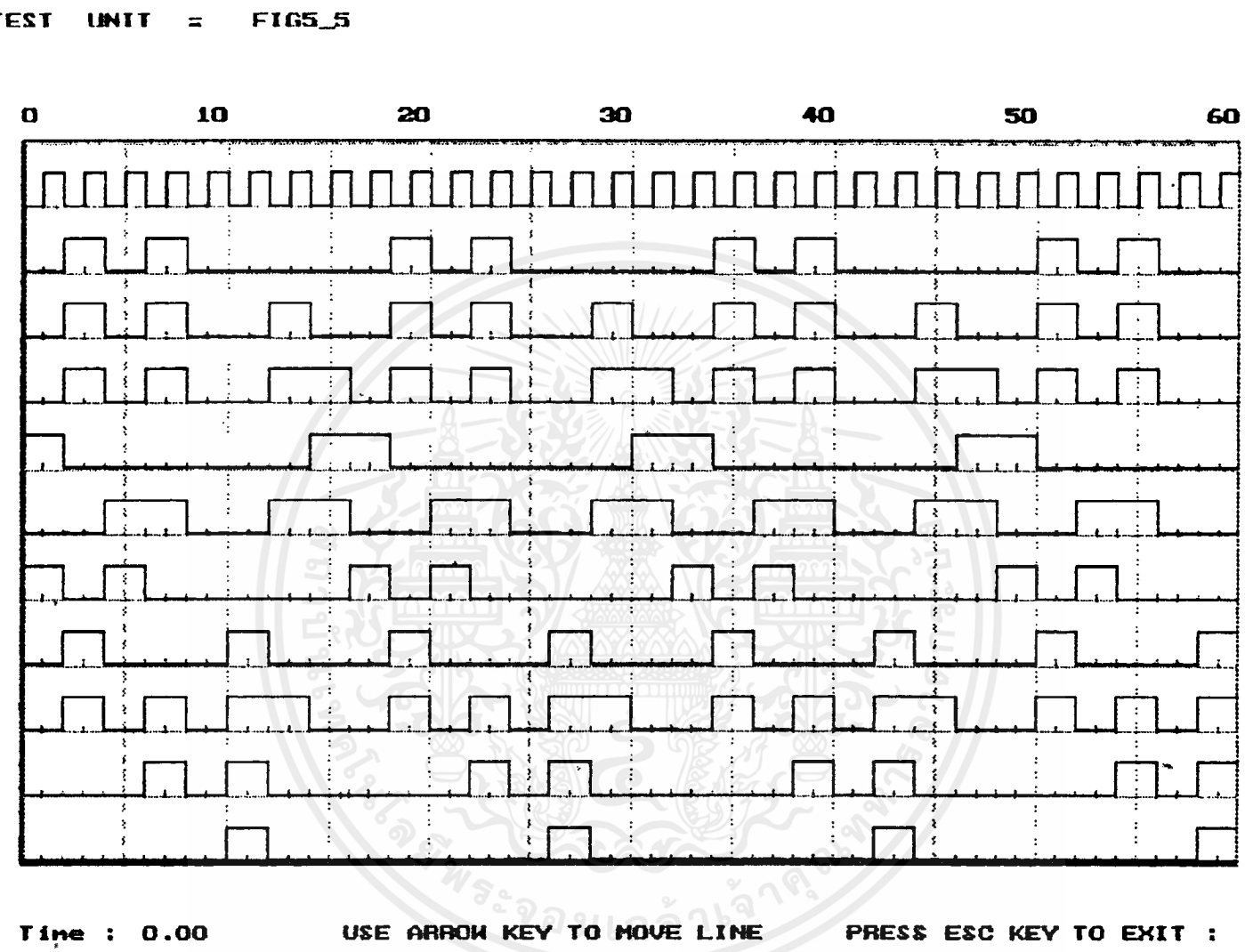
รูปที่ 5.3.2b แสดง Timing Diagram รูปที่ 5.3.2a



รูปที่ 5.3.3b แสดง Timing Diagram รูปที่ 5.3.3a



รูปที่ 5.4.3 แสดง Timing Diagram ของวงจรบวกเลข 8 bit



รูปที่ 5.5.1 แสดง Timing Diagram ของวงจร Synchronous Sequential

วิจารณ์และสรุปผล

โครงงานนี้มีวัตถุประสงค์เพื่อที่จะศึกษาการจำลองการทำงานของวงจรดิจิตอล โดยใช้กระบวนการทาง software ซึ่งมีประโยชน์อย่างมากในการที่จะนำมาใช้แทนการทดลองต่อวงจรจริง ผลที่ได้จากการทำงานของวงจรจะออกมาในรูป ไทม์ทิ่งไดอะแกรม (Timing Diagram) แสดงลอจิกของแต่ละ เอ้าท์พุท เทอร์มินัล (Output Terminal) ณ เวลาใดๆ

จากการทดลองในบทที่ 5 นำวงจรต่างๆมาทดลองทำการจำลองการทำงานของมัน ป้อนอินพุทต่างๆเข้าไป ผลลัพธ์ที่ได้ออกมาในรูป Timing Diagram ถูกต้องตรงกับความเป็นจริง

ในอนาคต หากได้มีการปรับปรุงให้มีความสามารถในการคิด delay ของอุปกรณ์แต่ละตัวด้วย จะทำให้มีประโยชน์มากขึ้น และเพื่อเพิ่มความสะดวกในการใช้งานอาจใช้การสั่งงานแบบเมนูก็จะทำให้สะดวกยิ่งขึ้น

หนังสืออ้างอิง

1. DONNA MQSICH, NAMIR SHAMMAS, BRYAN FLAMIG,"Avanced Turbo C Programmer's Guide", 1988

2. MILLERand QUILICI,"C Programming Reference: An AppliedPerspective"

3. วศิน เพิ่มทรัพย์, "คู่มือ MS-DOS PC-DOS", 2532

กิตติกรรมประกาศ

ในการทำปริญญานิพนธ์นี้ สำเร็จลงได้ด้วยความช่วยเหลือเป็นอย่างดียิ่งจาก อาจารย์ มนัส สังวรศิลป์ ซึ่งเป็นอาจารย์ที่ปรึกษาตลอดการทำปริญญานิพนธ์นี้ ท่านได้กรุณาช่วยเหลือ แนะนำ และให้ยืม เครื่องมือ เอกสาร และตำราต่างๆที่เกี่ยวข้องกับการทำปริญญานิพนธ์ พวกกระผมจึงต้องขอขอบคุณอาจารย์เป็นอย่างสูง และถ้ามีความผิดพลาดประการใดต้องขออภัยไว้ ณ ที่นี้ด้วย